UNIVERSIDAD PÚBLICA DE NAVARRA
NAFARROAKO UNIBERSITATE PUBLIKOA

TESIS DOCTORAL

UN ALGORITMO DISTRIBUIDO DE DETECCIÓN Y RESOLUCIÓN DE INTERBLOQUEO CON COSTE LINEAL Y CARÁCTER DINÁMICO.

María Antonia Castillo Latorre

DIRECTORES:

Dr. Federico Fariña Figueredo

Dr. Alberto Córdoba Izaguirre

A los que me han querido y ya no lo pueden hacer.

A los que me quieren sin pedir que yo les quiera.

Tabla de Contenidos

Resumen

En esta tesis se presenta un algoritmo para la detección y resolución de interbloqueos en sistemas distribuidos para el modelo de solicitud único recurso. El algoritmo resuelve todos los interbloqueos que aparecen en el sistema con coste $\mathcal{O}(n)$ en número de mensajes, siendo n el número de procesos interbloqueados. Una de las características más notables del algoritmo propuesto es que se adapta a las modificaciones del medio (aparición y desaparición de esperas entre procesos) sin que ello incremente el coste de la computación. Dotar de dinamismo a la solución propuesta supone un claro avance porque facilita su aplicación a escenarios cambiantes como lo es cualquier sistema en que se quiera resolver los interbloqueos detectados.

Además, en este trabajo e investigación se analizan las similitudes que existen entre el problema de elección de líder y el de detección y resolución de interbloqueo en sistemas distribuidos. En realidad, se puede decir que la resolución de interbloqueo distribuido es una versión dinámica del problema de elección de líder. La relación entre ambos problemas se mantiene cuando se analizan sus soluciones. Los algoritmos *edge-chasing* para resolución de interbloqueo pueden verse como versiones dinámicas de ciertos algoritmos de elección de líder en un anillo. Esto es cierto incluso si se considera la complejidad de ambos tipos de algoritmos. Los de interbloqueo mantienen la complejidad cuadrática de los algoritmos de elección de líder cuando trabajan en una configuración estática del sistema. En base a esta similitud, se disea un nuevo algoritmo que adapta a comportamientos dinámicos un algoritmo de elección de líder de complejidad lineal.

La tesis estudia diferentes mecanismos que son necesarios para conseguir que el algoritmo actualice su conocimiento de las esperas del sistema, a la vez que trata de localizar la forma de resolver un interbloqueo. Algunos de ellos, como el uso de 'memoria', ya se han utilizado en trabajos previos. Otros, como la simulación de identidad de nodos que han abandonado la ejecución, son novedosos y pudieran ser de utilidad en la adaptación de otros algoritmos a sistemas dinmicos.

La corrección del algoritmo se demuestra formalmente utilizando como herramienta la teoría de Autómatas de Entrada/Salida. La demostración comprueba que el algoritmo resuelve en un tiempo finito todos los interbloqueos que aparecen en el sistema y sólo detecta un interbloqueo si éste realmente existe. En el análisis de complejidad, a diferencia de lo habitual, se han considerado todos los mensajes que el algoritmo genera. No sólo se tienen en cuenta aquéllos que permiten la detección de un interbloqueo ya formado y los necesarios para evitar falsas detecciones, sino que también se contabilizan los mensajes que se generan durante el proceso de formacin del interbloqueo.

Agradecimientos

El trabajo que aquí se presenta es el fruto de un gran esfuerzo. Posiblemente, el lector de esta memoria nunca llegue a ser consciente del tiempo que le he dedicado y, mucho menos, de las versiones del algoritmo que se han ido descartando hasta exponer la que aparece seguidamente. La sensación de estar trabajando en un imposible ha sido el sentimiento que más me ha acompañado durante estos años. La verdad es que no sé qué extraña fuerza me ha impedido no tirar la toalla. Mi único deseo, ahora que doy por terminado el trabajo, es que a alguien le sirvan de algo todas estas páginas.

Y como dice el refrán que *es de bien nacido ser agradecido*, a continuación, quiero mostrar mi gratitud a:

- Dr. Federico Fariña Figueredo y Dr. Alberto Córdoba Izaguirre, directores de esta tesis. Gracias tanto por sus comentarios y correcciones como por su interés y su paciencia.
- El Grupo de Sistemas Distribuidos. Esta tesis ha pretendido mantener viva una de las líneas de investigación de este grupo en su origen. En el año 1977 los componentes más veteranos de este grupo iniciaron el estudio del tema del interbloqueo de procesos y muchos de los trabajos que se han venido desarrollando desde esa fecha han dado lugar a varias tesis doctorales. Espero que esta tesis sea digna merecedora de acompañar a todas ellas.
- Los compañeros del Área de Lenguajes y Sistemas Informáticos con los que compartí docencia desde el curso 2001-2002 hasta el curso 2009-2010 al mismo tiempo que me embarcaba en la aventura de la realización de la tesis.
- Los miembros del Departamento de Ingeniería Matemática e Informática y a los del antiguo departamento de Matemática e Informática. Estos dos departamentos respaldaron mis estudios de tercer ciclo.

- Los responsables, y resto de investigadores, de los proyectos de investigación en los que he figurado durante mi pertenencia al Grupo de Sistemas Distribuidos.
- La Facultad de Ciencias Económicas y Empresariales de la Universidad de Navarra que me ha permitido seguir vinculada a la docencia y al espíritu universitario en los años finales de este trabajo. Quiero mencionar especialmente a los profesores del Área de Métodos Cuantitativos del Departamento de Economía porque siempre me han animado a concluir esta etapa de mi formación académica.

Pamplona, 29 de Mayo de 2017

Capítulo 1

Introducción

El primer capítulo de esta memoria es una presentación general del problema del interbloqueo. En primer lugar, se proporcionarán diversas definiciones algo informales y generales del concepto de interbloqueo para luego, desde una perspectiva más rigurosa, llegar a explicar el problema que aquí se aborda. Analizar el interbloqueo de una manera formal conlleva identificar las condiciones que debe cumplir un sistema para que este problema aparezca. Una vez conocidas esas condiciones, se explicarán las tres técnicas clásicas desarrolladas para hacer frente a las situaciones de interbloqueo que pueden generarse en un sistema.

Además de estudiar los condicionantes para la existencia de un interbloqueo y los métodos ideados para su gestión, este capítulo proporcionará una clasificación de los sistemas atendiendo a los diferentes modelos de solicitud de recursos que en ellos pueden darse. Según el modelo de solicitud de recursos que adopte un sistema, la caracterización del interbloqueo será diferente y, por tanto, la forma de identificarlo y tratarlo será distinta.

Para completar esta introducción, se dedicará una sección a una revisión cronológica en los distintos ámbitos de aplicación donde se localiza el problema del interbloqueo, junto con las soluciones propuestas más destacadas. Dado que el algoritmo que se ha diseñado en esta tesis forma parte de un conjunto de soluciones concreto, se

hará especial hincapié en los algoritmos de este conjunto, esto es, para los algoritmos distribuidos de detección y resolución de interbloqueos con sistemas de modelo de solicitud único recurso.

El capítulo concluye citando una serie de trabajos que ponen de manifiesto que el problema del interbloqueo sigue siendo una tema de investigación.

1.1. Interbloqueo: definición y conceptos relacionados

Dado que el fin último de esta tesis es aportar una nueva solución al problema del interbloqueo, es conveniente que inicialmente se disponga de una idea intuitiva del concepto de interbloqueo. Aunque este término no está registrado como entrada en el Diccionario de la Real Academia Española, se puede adivinar su significado considerando que el prefijo *inter* completa el significado de la palabra *bloqueo*. Entre las acepciones de *bloqueo* se puede encontrar una que dice así: *acción y efecto de bloquear, esto es, de impedir el funcionamiento de una actividad*. Al incorporar el significado del prefijo a esta definición, se asume que este efecto ha sido producido por varios elementos/agentes o afecta a diversos elementos.

La palabra interbloqueo surge como traducción al castellano del término inglés *deadlock*, utilizado en el campo de la informática, entre otros. También es posible encontrar otras traducciones en castellano para referirse al mismo concepto, tales como *bloqueo mutuo*, *abrazo mortal* o *enlace mortal*.

En el diccionario *Online Etymology Dictionary* se puede consultar la siguiente definición:

> **deadlock (n):** *complete standstill, from* ***dead****(adj), in its emphatic use + **lock** (n).*

Si se consulta la palabra *deadlock* en un diccionario que explica términos relacionados con la informática, *The Free On-line Dictionary of Computing*, se puede ya disponer de un significado más cercano al tema que aquí va a ser tratado.

> **deadlock definition:** *(parallel, programming). A situation where two or more* ***processes*** *are unable to proceed because each is waiting for one of the others to do something. A common example is a program waiting for output from a server while the server is waiting for more input from the controlling program before outputting anything. It is reported that this particular flavour of deadlock is sometimes called a 'starvation deadlock', though the term 'starvation' is more properly used for situations where a program can never run simply because it never gets high enough priority. Another common flavour is 'constipation', in which each process is trying to send stuff to the other but all buffers are full because nobody is reading anything. See* ***deadly embrace****. Another example, common in* ***database*** *programming, is two processes that are sharing some resource (e.g. read access to a* ***table****) but then both decide to wait for exclusive (e.g. write) access. The term 'deadly embrace' is mostly synonymous, though usually used only when exactly two processes are involved. This is the more popular term in Europe, while* ***deadlock*** *predominates in the United States. Compare:* ***livelock****. See also* ***safety property****,* ***liveness property****.*

Después de este análisis semántico, resulta evidente que al hablar de interbloqueo se debe pensar, en un principio, en un problema que impide que determinada acción progrese, siendo la causa de este hecho el funcionamiento individual de cada uno de las partes que a su vez colaboran o participan en la consecución de la acción que se ha visto paralizada.

A partir de estas ideas generales e intuitivas del problema que va a ser tratado, es

más fácil comprender definiciones más formales del problema del interbloqueo en el mundo de la informática. Antes de presentar estas definiciones, se debe advertir que existe una dificultad añadida en esta tarea, ya que para explicar el término en cuestión es necesario describir previamente el ámbito de aplicación donde surge el problema del interbloqueo. Este inconveniente ya se dejó entrever en el extracto recabado del diccionario de informática mostrado anteriormente.

Si se considera un sistema informático en el que los procesos que en él se ejecutan se encuentran en un estado de espera indefinida, sin llegar a alcanzar su estado deseable de terminación, se puede identificar esta situación como de bloqueo indefinido de procesos o, simplemente, interbloqueo. En esta línea Gardarin en [1] define el interbloqueo para sistemas de memoria compartida de la siguiente manera: *El interbloqueo se puede definir, en el ámbito de los sistemas operativos, como la situación que se alcanza cuando un grupo de recursos han sido ocupados por un grupo de procesos en un orden tal que los procesos dentro del grupo no pueden continuar su ejecución.* Otros autores como Coffman [2], Brinch [3] o Rypka [4] han sugerido otras definiciones del concepto de interbloqueo, pero la definición de mayor aceptación y, por tanto, la más referenciada es la de Isloor *et al.* [5] que dice: *Un conjunto de procesos está en situación de interbloqueo cuando todo proceso en el conjunto está esperando por recursos asignados a otros procesos en el mismo conjunto.*

Por último, la definición de interbloqueo también se puede extender a los sistemas distribuidos. En este tipo de sistemas se comparten recursos y los procesos pueden solicitar el uso de recursos tanto locales como remotos, en un orden tal que éstos puedan quedar bloqueados indefinidamente. Si se pretende describir el problema del interbloqueo en otro ámbito de aplicación, tan conocido en la actualidad como es el de las bases de datos, hay que pensar que estos sistemas poseen mecanismos que permiten a los usuarios acceder a ellas para consultar o modificar datos. Si el acceso a la base de datos se puede realizar de forma simultánea y en ausencia de controles, se

pueden producir actualizaciones erróneas o inconsistentes en la información de la base de datos. La incorporación del control de concurrencia en estos sistemas propiciará la aparición de interbloqueos entre las transacciones y los datos sobre los que operan.

1.1.1. Condiciones necesarias para la existencia de un interbloqueo

En el trabajo de Coffman [2] se definieron las condiciones necesarias que deben cumplirse simultáneamente para que una situación de interbloqueo pueda afectar a un sistema. Las cuatro condiciones propuestas por Coffman se resumen seguidamente.

- **Condición de exclusión mutua** (*Mutual Exclusion*): Los procesos deben hacer uso exclusivo de los recursos asignados a ellos. El sistema debe tener un conjunto de recursos que no pueden ser compartidos simultáneamente por los procesos, es decir, sólo un proceso puede utilizar uno de esos recursos cada vez. Si un proceso solicita un recurso del conjunto en exclusión mutua y éste está asignado a otro proceso, entonces el proceso que lo solicita debe esperar hasta que el recurso haya sido liberado.

- **Condición de retención y espera** (*Hold and Wait or Resource Holding*): Hay procesos que retienen recursos asignados a ellos mientras esperan que les sean asignados otros recursos. Debe haber un proceso que tenga asignado al menos un recurso y esté esperando a adquirir otros recursos adicionales que están asignados a otros procesos.

- **Condición de asignación de recursos no expropiativa** (*No Preemption*): Los recursos no pueden ser asignados a otro proceso mientras el proceso que los tiene asignado no finalice y los libere. Un recurso se dice que está ocupado cuando un proceso lo tiene asignado en exclusividad y lo puede utilizar. En caso

contrario, se dice que está vacío o libre. Vaciar un recurso asignado significa obtenerlo del proceso, guardando el estado del proceso y del recurso en ese instante y restaurando un estado inicial del recurso. La condición establece que los recursos no pueden ser prevaciados, es decir, un recurso sólo puede ser liberado voluntariamente por el proceso que lo tiene asignado, después de que haya completado su acción sobre el recurso.

- **Condición de espera circular** (*Circular Wait*): Un ciclo de procesos consiste en una cadena circular en la que cada proceso espera por uno o más recursos que, a su vez, son retenidos por el siguiente proceso de la cadena. Es decir, un conjunto de procesos $\{p_0, p_1, p_2,...,p_{n-1}\}$ tales que $\forall\ i$: $0 \ldots n-1$, $p_{i\ mod\ n}$ espera por un recurso asignado al proceso $p_{(i+1)\ mod\ n}$.

Este conjunto de condiciones no es mínimo ya que fácilmente se puede comprobar que las condiciones no son independientes. Por ejemplo, resulta evidente que la condición de espera circular implica la de retención y espera. Coffman no se planteó encontrar unas condiciones mínimas e independientes, sino encontrar una formulación práctica. Algunas de las técnicas desarrolladas para el tratamiento del interbloqueo consisten en evitar que se cumpla alguna de las condiciones apuntadas por Coffman (técnicas de prevención). Incluso cuando tienen lugar estas condiciones, determinar si va a existir un interbloqueo o cuándo ocurrirá, no es inmediato.

1.1.2. Grafos de Asignación de Recursos y Grafos de Esperas

Holt en sus trabajos [6] y [7] fue uno de los primeros investigadores en representar el interbloqueo con un modelo basado en teoría de grafos. Para ello, adaptó la información contenida en una tabla de asignación de recursos a un grafo con la finalidad de caracterizar después en esta representación gráfica la situación del interbloqueo.

El Grafo de Asignación de Recursos (*Resource Allocation Graph - RAG*) que se emplea habitualmente es un grafo bipartito, dirigido y dinámico, $G \equiv(V, E)$, donde V es el conjunto de nodos del sistema y E el conjunto de arcos que representan las relaciones entre los nodos. El conjunto de nodos V, por su parte, está formado por dos subconjuntos disjuntos $V = P \cup R$, siendo P el conjunto de procesos del sistema y R el conjunto de recursos. Los recursos son reutilizables y de uso exclusivo. El conjunto de arcos, E, verifica en todo instante que $E \subseteq V \times V$. Un arco, $e \in E$, es un par ordenado que indica que existe un arco dirigido desde un proceso a un recurso o viceversa, $e = (p_i, r_j)$ o $e = (r_j, p_i)$. En el primer caso, el arco indica que el proceso ha solicitado un recurso y se ha quedado esperando por él (arco de solicitud). Si, por el contrario, el arco parte del recurso hacia el proceso, entonces el arco indica que un recurso ha sido asignado al proceso (arco de asignación).

En un instante de tiempo t, habrá una configuración determinada de arcos, $E(t)$, en el grafo G. La actualización de los arcos es de vital importancia y la lleva a cabo el gestor de recursos y procesos del sistema. La operación de solicitar un recurso puede provocar la aparición de un arco de solicitud o de un arco de asignación. Sin embargo, la operación de liberar un recurso siempre tiene como efecto la desaparición del correspondiente arco de asignación y, en consecuencia, puede convertir ciertos arcos de solicitud en arcos de asignación. Por tanto, resulta obvio indicar que el grafo G tienen que estar actualizado, manteniendo así un estado consistente con la realidad del sistema.

A partir de la representación gráfica del grafo de asignación de recursos de un sistema, se muestra fácilmente que la existencia de un circuito en el grafo puede implicar o no una situación de interbloqueo, pero la existencia de un interbloqueo siempre está asociada a la presencia de un circuito en el grafo. Si los recursos del sistema son únicos, la existencia de un circuito en el grafo de asignación de recursos es condición necesaria y suficiente para que exista interbloqueo en el sistema. Esta

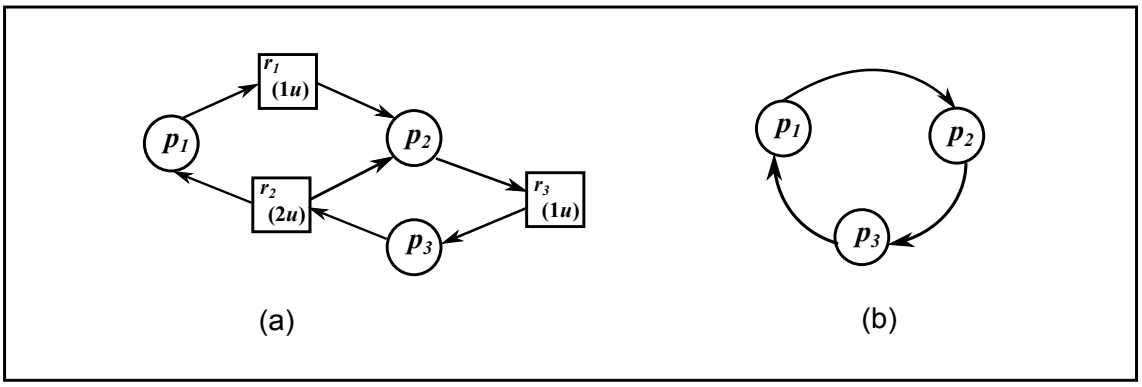

Figura 1.1: Grafo de asignación de recursos, *RAG* (a) y grafo de esperas relacionado, *WFG* (b).

caracterización del interbloqueo es también la considerada en [2] y en [8].

La simplificación del grafo de asignación de recursos deriva en el denominado Grafo de Esperas (*Wait-For Graph-WFG*). En esta versión del grafo los nodos sólo representan a los procesos. La existencia de un arco dirigido del proceso p_i al proceso p_j indica que el proceso p_i está esperando por algún recurso que tiene asignado el proceso p_j. Aunque trasladar el dinamismo de un sistema (aparición y borrado de esperas) resulta más complicado en un grafo de tipo *WFG* que en un *RAG*, el uso del grafo simplificado está más extendido, porque facilita la creación de estructuras de datos abstractas sobre las que los algoritmos de detección pueden mejorar su complejidad en tiempo de ejecución. La figura 1.1 muestra un ejemplo de un grafo de asignación de recursos y su correspondiente grafo de esperas.

1.1.3. Técnicas para el tratamiento del interbloqueo

Las estrategias básicas desarrolladas para el tratamiento del interbloqueo pueden clasificarse, según [2], en:

- **Técnicas de prevención:** Las medidas preventivas se adoptan para evitar que el interbloqueo llegue a aparecer en el sistema. El fundamento de la prevención

del interbloqueo es, por lo tanto, diseñar el sistema de tal forma que, en cada instante de tiempo, al menos una de las condiciones necesarias establecidas por Coffman no sea satisfecha.

- **Técnicas de exclusión (*avoidance*):** Consisten en la aplicación de unos protocolos alternativos que impiden que el interbloqueo tenga lugar. A diferencia de las técnicas de prevención el sistema no tiene que satisfacer ningún requisito *a priori* que impida la aparición de un interbloqueo. Para que el interbloqueo no ocurra en un sistema que opta por la técnica de exclusión, es fundamental que se disponga de información previa de las necesidades futuras del sistema. A partir de esta información se crean mecanismos interactivos para asegurar la ejecución de las tareas del sistema. La cantidad y el tipo de información requerida varía según el modelo usado por el algoritmo específico de exclusión de interbloqueo. Con este algoritmo se puede asegurar que, en un instante del tiempo, la condición necesaria de la espera circular no se satisface y el sistema permanece en un estado seguro. El problema de la exclusión se ha abordado considerando distintos enfoques: la imposibilidad de algunos esquemas de exclusión, la interpretación según la teoría de juegos y analizando las particularidades de los sistemas en los que se incorpora.

- **Técnicas de detección:** En muchas ocasiones, diseñar un sistema en el que nunca ocurra el interbloqueo es imposible. Por este motivo, es necesario que exista una solución que permita que el sistema evolucione libremente y, después de que el interbloqueo se produzca, se adopten medidas para recuperar el funcionamiento normal del sistema. Para que esta aproximación funcione correctamente, el sistema debe disponer de un algoritmo para detectar la aparición del interbloqueo y debe contar con algún mecanismo que resuelva el interbloqueo eliminándolo del sistema.

Después de haber descrito brevemente las tres técnicas de tratamiento del interbloqueo, se van a analizar más detenidamente cada una de ellas para así conocer las ventajas y los inconvenientes de su aplicación en un sistema.

1.1.3.1. Técnicas de prevención

Los mecanismos de prevención fueron introducidos por Havender [9]. En este trabajo se define un esquema de ordenamiento de recursos para prevenir el interbloqueo en el sistema operativo de IBM OS/360.

A continuación, se presentan distintos mecanismos de prevención según la condición de Coffman que se desea incumplir para eludir el problema del interbloqueo:

- *Exclusión mutua.* El incumplimiento de la condición de la exclusión mutua no se puede asegurar para todos los tipos de recursos. Muchos de los recursos de un sistema no pueden ser compartidos simultáneamente por varios procesos y, por lo tanto, en estos casos, evitar la exclusión mutua es prácticamente imposible. En cambio, los recursos que pueden ser compartidos por varios procesos, por ejemplo, los ficheros de sólo lectura, no requieren accesos exclusivos y nunca están implicados en situaciones de interbloqueo.

- *Retener y esperar.* La segunda de las condiciones de Coffman puede ser incumplida si a ningún proceso se le permite solicitar recursos mientras ocupe otro. Un modo de conseguir esto es forzar a que todos los procesos soliciten todos los recursos que van a utilizar antes de que su ejecución comience. Si es posible, se les asignarán todos los recursos que precisan y si no, los procesos no podrán iniciar su ejecución. Una alternativa a esta aproximación es obligar a que los procesos liberen todos los recursos ocupados antes de solicitar recursos adicionales. Estos dos protocolos presentan dos grandes desventajas. La primera de ellas es que la utilización de los recursos probablemente resulte baja,

porque los recursos podrían haber sido asignados a un proceso que no los use durante la mayor parte del tiempo. La segunda desventaja es que puede haber procesos que esperen indefinidamente por algunos recursos, que son utilizados con mucha frecuencia por un conjunto de procesos (*starvation*). Además, si un proceso requiere múltiples recursos, no hay garantía de que todos los recursos que necesita estén desocupados en un momento dado del tiempo. Si esto no ocurre, el proceso será forzado a esperar un periodo de tiempo de tamaño arbitrario. En cualquier caso, la limitación de la concurrencia lleva consigo un bajo rendimiento del sistema.

- *Asignación de recursos no expropiativa.* Para evitar que se cumpla la tercera condición basta con que se requisen los recursos que ya hayan sido asignados a un proceso. Se puede usar el siguiente protocolo. Cuando un proceso p solicita recursos que no pueden ser inmediatamente asignados a él, es decir, el proceso debe esperar, entonces todos los recursos asignados a p son inmediatamente requisados. Por tanto, en algún instante futuro cuando todos los recursos previamente ocupados y nuevamente solicitados por el proceso p estén disponibles, p puede seguir la ejecución. Como alternativa podría emplearse otro protocolo que establece que, cuando un proceso solicita recursos, se haga una comprobación que asegure que los recursos estén disponibles [10]. Si los recursos concretos están disponibles, podrán ser asignados al proceso solicitante. En cambio, si otro proceso que los retiene está ocupando los recursos precisos para el proceso solicitante, los recursos se requisarán al proceso retenedor. El proceso que espera tendrá que conseguir los recursos requisados antes de que pueda continuar con su ejecución. Si los recursos no están disponibles ni retenidos por un proceso que espera, el proceso solicitante debe esperar. Durante la espera, los recursos retenidos por el proceso solicitante podrían ser requisados si otro proceso los

solicita. Este protocolo sólo puede usarse con recursos cuyo estado pueda ser fácilmente guardado y recuperado. Recursos tales como el procesador o la memoria principal pueden ser vaciados sin una gran carga para el sistema. Por supuesto, no todos los recursos verifican este criterio. Además, podría no ser siempre práctico guardar y recuperar el estado de los recursos, especialmente si el sistema tiene límites computacionales. Esta técnica impone un bajo rendimiento del sistema puesto que se deben suspender operaciones sobre los recursos ya iniciadas para luego volver a retomarlas.

- *Espera circular*. La condición final para el interbloqueo puede ser violada si se impone una ordenación total en el acceso al conjunto de todos los recursos. Se necesita que cada proceso solicite recursos sólo en un orden creciente de enumeración. Por ejemplo, considérese un sistema con los siguientes recursos $R = \{r_1, r_2,...r_n\}$. La condición de espera circular se incumple si todos los procesos solicitan recursos sólo en un orden creciente. Inicialmente, un proceso puede solicitar cualquier cantidad de r_i. Después de eso, el proceso puede solicitar recursos r_j si y sólo si el número asignado a r_j en la ordenación total es estrictamente mayor que la asignada a r_i. Por ejemplo, si el proceso p necesita tanto r_1 como r_3 para una ejecución completa, debe solicitar los recursos exactamente en ese orden. En este método se puede comprobar el ordenamiento de los recursos antes de la ejecución de los procesos, o sea, en el tiempo de compilación [11]. Además, el orden de los recursos se puede establecer de manera natural como en [12] y en [13]. A pesar de todo ello, en algunos sistemas, como los de bases de datos, no se puede aplicar un ordenamiento global *a priori* para todos los tipos de recursos.

Las técnicas de prevención son sencillas de aplicar, pero afectan directamente a la concurrencia de los procesos que ejecuta un sistema, restringiendo su utilidad y disminuyendo el rendimiento del sistema.

1.1.3.2. Técnicas de exclusión

Consisten en la aplicación de protocolos alternativos a la prevención, y a la detección y resolución de interbloqueos. El interbloqueo no debe ocurrir, pero no se obliga al sistema a satisfacer ciertos requisitos *a priori* como en las técnicas de prevención. El aspecto crucial para que la exclusión de interbloqueos sea un éxito es que se avance alguna información, indicando las necesidades de recursos de los procesos y cómo los procesos solicitarán recursos en el futuro. Los algoritmos de exclusión interactúan con las operaciones de ocupación de recursos y no limitan la forma de realización de las mismas. La información puede ser usada para determinar si, al atender cada solicitud, la asignación de recursos situará al sistema en un estado posible de interbloqueo. Los métodos que comprueban si en la ejecución de procesos se puede producir un interbloqueo, no son propiamente técnicas para el tratamiento del interbloqueo. Con la información previa habrá un mecanismo interactivo para que todos los procesos puedan terminar su ejecución. La cantidad y el tipo de información requerida varía dependiendo del modelo usado por cada algoritmo de exclusión de interbloqueo. El modelo más sencillo hace uso del peor caso de los requerimientos de recursos para cada solicitud. Dada la información del modelo, un algoritmo de exclusión de interbloqueo puede asegurar que, en un instante del tiempo, la condición necesaria de espera circular para el interbloqueo no se satisface y el sistema permanece en un estado seguro.

Comparando las técnicas de exclusión con las de prevención, se puede concluir que las últimas limitan fuertemente la concurrencia de las ejecuciones de las tareas del sistema. Sin embargo, las primeras pueden llegar a ser más eficaces si la información relativa a las necesidades para el funcionamiento del sistema no es muy variable.

1.1.3.3. Técnicas de detección

Como se ha visto, diseñar un sistema en el que el interbloqueo nunca pueda ocurrir no es factible, ya que conduce a un sistema de bajo rendimiento. Otra solución puede consistir en que se permita que el interbloqueo tenga lugar y, después de que ocurra, se tomen medidas para recuperar el funcionamiento normal del sistema. Para que esta aproximación funcione correctamente, el sistema debe disponer de un algoritmo para detectar el interbloqueo cuando aparezca y de un mecanismo que resuelva el interbloqueo eliminándolo del sistema.

El algoritmo de detección se ejecuta periódicamente para comprobar si el estado del sistema se corresponde con una situación de interbloqueo. Generalmente, el estado del sistema se corresponde con la información existente de las operaciones que realizan los procesos sobre los recursos y que pueden quedar representadas en el grafo de asignación de recursos. El algoritmo de detección tiene como objetivo encontrar circuitos en dicho grafo. Para un correcto funcionamiento del algoritmo es necesario mantener actualizada la información sobre la asignación de recursos y sobre las peticiones pendientes. Si se detecta un interbloqueo, se requiere de un mecanismo de recuperación para que algunos de los procesos involucrados puedan continuar su ejecución. En consecuencia, el coste en la detección será el tiempo que se precisa para encontrar un interbloqueo y, además, se deberá considerar un coste adicional de las tareas concretas de recuperación del interbloqueo. En muchos casos la tarea de recuperación es más laboriosa que el propio mecanismo de la detección.

Para sistemas donde los recursos tienen una única instancia, un interbloqueo existe en el sistema si y sólo si existe un ciclo en el grafo de esperas. Su detección precisa que el sistema esté monitorizando las asignaciones de recursos en el grafo de esperas y periódicamente invoque un algoritmo que chequee la presencia de un ciclo en el grafo de esperas. Sin embargo, esta forma de detección es insuficiente para sistemas en los que los recursos puedan tener múltiples instancias. Para tales sistemas, se precisa una

aproximación más complicada. El algoritmo tendrá que utilizar varias estructuras de datos que controlen la cantidad de recursos disponibles, la asignación de recursos a cada proceso y la cantidad de recursos que cada proceso está solicitando. Cuando el algoritmo es invocado, usa estas estructuras de datos para examinar cada secuencia posible de asignación en los procesos, que han sido iniciados pero todavía no se han completado. Si no pueden ser terminados todos los procesos iniciados, queda confirmada la existencia de un interbloqueo. La mayor preocupación, cuando se utiliza un algoritmo de detección, es la decisión de las condiciones en que se debe ejecutar el algoritmo de detección. Si la naturaleza del sistema es tal que los interbloqueos ocurren frecuentemente, el algoritmo de detección debería ser invocado frecuentemente. Otro factor que debería ser considerado es la sobrecarga requerida para ejecutar el algoritmo. Si es extremadamente alta, ejecutar el algoritmo de detección muy a menudo podría llegar a ser inaceptable.

Métodos de recuperación

Los algoritmos de detección de interbloqueos no son suficientes por sí solos para resolver el problema del interbloqueo. Un algoritmo de detección debe acompañarse de algún medio de recuperación cuya función sea librar al sistema de todos los interbloqueos. Hay dos mecanismos básicos de recuperación:

- En la primera opción de recuperación habitual, se para la ejecución de un proceso de los que están interbloqueados y se vacían los recursos que tiene asignados. Los recursos así liberados se reasignan a otros procesos que estaban interbloqueados y esperaban por esos recursos para proseguir su ejecución. Por su parte, el proceso que interrumpió su ejecución para romper el interbloqueo puede restaurar su estado y reiniciar desde algún estado conocido su ejecución. Este método no se puede aplicar en todas las ocasiones porque requiere el prevaciado de recursos y esto no siempre es posible por el coste que supone para el sistema.

- El segundo método consiste en abortar la ejecución de uno o más procesos implicados en el interbloqueo. Los recursos obtenidos de esta manera son asignados a otros procesos involucrados en el interbloqueo para que puedan proseguir su ejecución. La elección del proceso o procesos que deben ser abortados fue estudiada por Coffman en [2], considerándola un problema de optimización. Así pues, el proceso que debe ser abortado es aquel cuyo coste sea mínimo para el sistema. La función objetivo que se desea optimizar puede ser:

  - La prioridad de ejecución del proceso.
  - El tiempo de ejecución ya consumido por el proceso junto con una estimación del tiempo que le resta para terminar su ejecución.
  - Los tipos de recursos que el proceso está utilizando en su ejecución.
  - El número de recursos que el proceso necesita para concluir su ejecución.
  - El número de procesos a los que se les podría asignar los recursos que quedarían liberados si el proceso fuera el elegido para ser abortado.
  - El número de veces que el proceso elegido para ser abortado ha sido seleccionado en otros interbloqueos anteriores.

1.1.4. Clasificación de modelos de computación

Una forma muy extendida de clasificar los sistemas en los que puede aparecer el problema del interbloqueo se basa en los tipos de petición de recursos que pueden realizar los procesos. El tipo de petición de recurso en un sistema puede tener relación con la aplicación que se vaya a dar de él. Knapp [14] estableció una clasificación de los diferentes modelos de solicitud de recursos. En esta clasificación se considera que todos los recursos son de uso exclusivo, permitiendo a su vez comparar los algoritmos según la complejidad de los sistemas donde se pueden implementar.

- **Modelo único recurso (*Single Unit Resource Request Model-SR*).** Es el modelo de petición de recursos más simple y el que se suele usar en estudios teóricos. Los procesos solicitan los recursos que necesitan de uno en uno, de manera que en un instante de tiempo cada proceso sólo está esperando por un único recurso. Considerando un grafo como la representación gráfica de las relaciones entre procesos y recursos, se observa que de un nodo únicamente puede partir un arco. Por consiguiente, un interbloqueo en este modelo se caracteriza por un ciclo en el grafo que representa las relaciones de espera [14].

- **Modelo *AND*.** La solicitud de recursos en este modelo se realiza por conjuntos. Cuando se produce la petición de un conjunto de recursos por parte de un proceso, éste permanecerá bloqueado hasta adquirir la totalidad de los recursos solicitados. En un grafo donde se representan las relaciones de espera, de cada nodo del sistema puede salir más de un arco. La presencia de un interbloqueo en un sistema con modelo *AND* sigue caracterizándose por la existencia de un ciclo en el grafo de esperas [14]. Aun siendo ésta la condición necesaria y suficiente para confirmar la existencia de un interbloqueo, la estructura del mismo podrá adoptar formas más complejas [15].

- **Modelo *OR*.** En este modelo los procesos también solicitan los recursos necesarios por conjuntos. A diferencia del modelo *AND*, después de cada petición de recursos, el proceso permanece bloqueado tan sólo hasta que se le asigna alguno de los recursos solicitados. La caracterización de un interbloqueo en el modelo *OR* consiste en la localización de un nudo en el grafo de esperas [8]. La existencia de un ciclo es una condición necesaria, pero no suficiente, para determinar que hay un interbloqueo en el sistema [16].

- **Modelo *M de N*.** Este modelo de petición de recursos implica que los procesos realizan peticiones por N recursos y se mantienen bloqueados hasta que

adquieren M recursos de los solicitados. Aunque la teoría de grafos no permite describir la existencia de un interbloqueo para este modelo mediante una estructura simple, la localización de un ciclo no será suficiente pero sí necesaria [17].

- **Modelo general.** Los procesos, en este modelo, realizan sus peticiones de recursos según la definición de una función lógica sobre el conjunto de recursos. Tras una petición de recursos, un proceso queda esperando hasta que los recursos que se le han asignado verifiquen la función lógica. En el modelo general, la existencia de un interbloqueo no se corresponde con ninguna estructura simple de las descritas en la teoría de grafos. A pesar de eso, un ciclo será una condición necesaria, aunque no suficiente, para la existencia de un interbloqueo en el sistema.

Knapp también define un modelo llamado *AND-OR* que no se ha incluido aquí por coincidir con el modelo general descrito. Es obvio que el modelo de único recurso es un caso particular del modelo *AND* y del modelo *OR*. Además, resulta evidente que estos últimos modelos son casos particulares del modelo *M de N* y éste a su vez del modelo general.

1.2. Revisión histórica del problema del interbloqueo en sistemas de memoria compartida

Se puede decir, sin riesgo a que nadie lo ponga en duda, que el problema del interbloqueo fue descubierto por Dijkstra cuando realizaba pruebas al sistema operativo IBM-360 [18]. Inicialmente el interbloqueo fue analizado en sistemas multiprogramados y, posteriormente, las investigaciones se extendieron a los protocolos de

comunicación de los sistemas de bases datos y al área de los sistemas distribuidos.

1.2.1. Soluciones al interbloqueo en sistemas operativos

Entre 1960 y 1970 aparecieron los primeros trabajos centrados en el interbloqueo en sistemas con memoria compartida. Cabe destacar las contribuciones de: Dijkstra en [18] y [19], Havender en [9], Murphy en [20] y Habermann en [21]. Sus investigaciones se centraban en sistemas multiprogramados. En este tipo de sistemas, los procesos compiten por la utilización de un conjunto finito de recursos. Un proceso equivale a la ejecución de un programa secuencial. Un recurso es cualquier componente del sistema necesario para la ejecución de los distintos procesos: la memoria principal y secundaria, el procesador, los dispositivos físicos, loa circuitos de entrada/salida y la información contenida en algunos registros. Los procesos deben adquirir los recursos antes de utilizarlos y deben liberarlos después de utilizarlos. La secuencia de operaciones sobre un recurso en un funcionamiento normal consiste en: la solicitud del recurso (llamada al sistema operativo), la utilización del recurso y, finalmente, la liberación del recurso (llamada al sistema operativo). El gestor de recursos del sistema operativo es el encargado de mantener actualizada la información sobre los recursos disponibles y asignados, además de una lista con los procesos que esperan por cada recurso asignado.

Los procesos entran en estado de espera después de solicitar un recurso que no se encuentra disponible. De esta forma, un grupo de procesos puede quedar interbloqueado. La aparición de un interbloqueo no sólo depende de la corrección de los programas, sino que también influyen las características del sistema [22]. Aunque se empleen técnicas de programación que disminuyan la probabilidad de aparición de interbloqueos, no se evita el problema. Por otra parte, los recursos del sistema pueden ser de muy diversos tipos, según su naturaleza (permanentes/temporales en [6]), las operaciones que soportan, el número de unidades disponibles (atómicos/M

de N en [23]) y las restricciones de su uso. En consecuencia, es más que conveniente particularizar el análisis del interbloqueo al tipo de recurso que el sistema considera.

Dado que la mayoría de los sistemas operativos presentan una estructura jerárquica o de niveles de programación, que da lugar a un ordenamiento de los recursos por esos niveles, las técnicas de prevención, exclusión y detección se suelen combinar para obtener una solución completa al problema del interbloqueo en ellos. Howard en [12] presenta un método en el que considera las tres técnicas descritas en la sección 1.1.3. Hasta bien entrados los años 80, surgieron muchas aportaciones para tratar el interbloqueo en sistemas de memoria compartida. Los trabajos de [9] y [12] proporcionan ejemplos de técnicas de prevención para sistemas operativos. Las investigaciones de [8], [21] y [24] entre otras son ejemplos de algoritmos de exclusión. El problema de la exclusión se ha abordado desde distintas perspectivas: considerando la imposibilidad de algunos esquemas de exclusión en [25] y [26], proporcionando una interpretación de teoría de juegos [27] y analizando el problema, de acuerdo a las particularidades de otros tipos de sistemas como los de bases de datos y los sistemas distribuidos, [28] y [29]. El esquema de exclusión más utilizado y conocido es *el algoritmo de los banqueros* propuesto por Dijkstra [19] para un sistema operativo con recursos de tipo *M de N*. En cuanto a algoritmos detección y resolución propios de sistemas centralizados, destacan los de [2] y [30].

1.2.2. Soluciones al interbloqueo en bases de datos

Para el tratamiento del interbloqueo en bases de datos se utilizan básicamente los mismos métodos que en sistemas operativos: técnicas preventivas, técnicas de exclusión y técnicas de detección y resolución de interbloqueos.

Se han llegado a proponer soluciones que siguen *estrategias preventivas* que evitan la ocupación permanente de los recursos (atributos en sistemas de bases de datos). Controlando el acceso de las operaciones o transacciones que se realizan sobre los

datos, es posible evitar que las transacciones puedan quedar bloqueadas indefinidamente. Otras soluciones planteadas para sistemas operativos, con el fin de prevenir las situaciones de interbloqueo, carecen totalmente de sentido en sistemas de bases de datos por las características propias de los recursos/atributos que se almacenan y tratan en ellos. Así pues, no es posible establecer un orden entre los atributos que evite las esperas circulares (cuarta condición de Coffman) o imponer una ocupación *a priori* de todos los atributos necesarios para impedir la ocupación parcial de estos recursos (tercera condición de Coffman).

Por otra parte, los *algoritmos de exclusión* en bases de datos centralizadas, al igual que los desarrollados para sistemas operativos, planifican las transacciones, teniendo en cuenta los diferentes tipos de atributos que se pueden utilizar. Como estos algoritmos deben garantizar que el sistema pase de un estado seguro a otro sin que existan interbloqueos, es necesario que se compruebe esta circunstancia en todo momento. Para que la ejecución de este test de seguridad no suponga un coste elevado para el sistema, se suelen implementar mecanismos que hacen que sólo se ponga en marcha en situaciones concretas. Por ejemplo, en [31], cuando las transacciones intentan ocupar recursos/atributos no incluidos en su declaración de necesidades o cuando se topan con otras transacciones bloqueadas al intentar ocupar un determinado recurso.

Si se consideran los *tratamientos de detección y recuperación* de interbloqueos en bases de datos, es necesario, a su vez, conocer los modelos de ocupación que pueden admitir las bases de datos. En consecuencia, se podrán encontrar algoritmos de detección para las distintas formas de ocupación de recursos. La clasificación de modelos de ocupación establecida en [14] se puede adaptar fácilmente (sustituyendo proceso por transacción) al ámbito de aplicación que se está analizando, esto es, a un esquema transaccional de operaciones sobre una base de datos. Es importante señalar que un algoritmo de detección válido para un modelo de ocupación de complejidad superior se puede utilizar para un modelo de complejidad inferior.

Además de la dificultad que puede suponer construir un algoritmo de detección de interbloqueo para bases de datos con un modelo de ocupación más o menos complejo, las características y funcionalidades propias de un sistema de bases de datos provocan un incremento de esa dificultad. Destacan, por ejemplo, que los nombres de los recursos no sean únicos, que los recursos puedan cambiar de categoría después de una operación (categorías dinámicas), que la ocupación de un recurso dependa de las ocupaciones previas de otros recursos (ocupaciones interdependientes) o que, para lograr la máxima concurrencia de procesos accediendo a la base de datos, la ocupación de un recurso conlleve realmente la ocupación de una infinidad de recursos que lo componen (incremento en el orden de magnitud del número de ocupaciones). En consecuencia, aunque se utilicen los mismos métodos para estudiar el interbloqueo en sistemas operativos y en bases de datos, el estudio del problema en bases de datos resulta mucho más complicado por la naturaleza de sus recursos [32].

En [8] y en [33] se representan situaciones de interbloqueo en bases de datos mediante grafos de esperas. En estos grafos dirigidos, los nodos simbolizan transacciones y los arcos indican relaciones de espera entre transacciones. Si la transacción T_i está esperando para ocupar alguno de los recursos que posee la transacción T_j, en el grafo habrá un arco que parta de T_i y llegue hasta T_j. Es obvio, por tanto, que la presencia de interbloqueos en el sistema de base de datos se caracterizará también por la existencia de ciclos en el grafo de esperas que se obtiene al representar las operaciones de solicitud de recursos del sistema. Una vez que el algoritmo de detección ha cumplido su objetivo, el sistema debe encargarse de subsanar la situación para poder funcionar de nuevo correctamente. Entre las opciones implementadas más frecuentemente destacan las que deshacen los efectos de una o más transacciones. Cuando se adopta este tipo de solución es fundamental elegir adecuadamente las transacciones (víctimas) que tienen que echar marcha atrás en su ejecución para

recuperar el funcionamiento normal de la base de datos. Para determinar qué transacciones interbloqueadas van a permitir romper la situación de interbloqueo, se deben analizar varios factores: el tiempo que ha transcurrido desde que se inició la ejecución de la transacción y el tiempo que le resta para su finalización, el número de atributos/recursos que ha utilizado la transacción hasta el momento de la detección, la cantidad de atributos/recursos que aún necesita hasta finalizar y, por último, cuántas transacciones se verán afectadas en el retroceso de la transacción elegida como víctima. Cualquier decisión que se adopte tratará de que la resolución del interbloqueo se haga con el menor coste posible, primando el factor o los factores más críticos para el sistema y la aplicación que desarrolla.

Tras escoger la transacción o transacciones que van a retroceder en su ejecución, debe considerarse si el retroceso será total o parcial. Aunque la primera opción es la más drástica, porque supone suspender la ejecución de la transacción y tener que volver a iniciarla, es también la más sencilla de realizar. El retroceso parcial, sin embargo, es un procedimiento más efectivo porque la transacción retrocede justo lo necesario para acabar con el interbloqueo. Como contrapartida, el retroceso parcial requiere que el sistema disponga en todo momento de información adicional actualizada sobre el estado de las transacciones activas del sistema. Los algoritmos que implementan esta forma de resolver las situaciones de interbloqueo también suelen incorporar mecanismos que eviten que la transacción, seleccionada como víctima, sea siempre la misma. En los sistemas reales es muy importante garantizar que todas las transacciones podrán ser ejecutadas (principio de equidad).

1.3. Revisión histórica del problema de interbloqueo en sistemas distribuidos

Con motivo del desarrollo de los sistemas distribuidos, el problema del interbloqueo volvió a ganar importancia. Un sistema distribuido está formado por un conjunto de lugares (entidades) que pueden procesar y almacenar información de forma independiente, pero que requieren de una red de comunicaciones para intercambiar información mediante mensajes. Una ventaja que presentan los sistemas distribuidos y a la que deben en gran medida su expansión es la posibilidad de compartir recursos [34]. Así pues, desde cualquier ubicación del sistema se puede acceder a cualquiera de sus recursos por muy alejados que estén. Es precisamente el hecho de que los procesos puedan solicitar el uso de recursos (locales o remotos) en un orden determinado lo que puede hacer que queden bloqueados indefinidamente y aparezca, por tanto, una situación de interbloqueo.

En la literatura de sistemas distribuidos, el interbloqueo aparece frecuentemente y, además, en campos muy diversos: protocolos de comunicaciones [35], [36], [37], programación en *ADA* [38], redes de Petri [39], encaminamiento de mensajes [40], [41], [42], [43], [44], [45], técnicas de simulación [46], sistemas de robots distribuidos [47],... Los primeros trabajos sobre interbloqueo en sistemas distribuidos se centraron tanto en interbloqueos de recursos como en interbloqueos de comunicaciones. Las definiciones de ambos tipos de interbloqueo son confusas y las diferencias entre ellos no resultan muy claras. El interbloqueo de comunicaciones requiere que, dado un conjunto de procesos, cada proceso del conjunto espere por un mensaje de algún otro proceso del mismo conjunto y, en definitiva, ningún proceso del conjunto envía mensajes. Aunque a lo largo de este trabajo se hace referencia básicamente al interbloqueo de recursos, el estudio realizado es válido en cualquier ámbito de aplicación.

El desarrollo de los sistemas distribuidos se atribuye normalmente a dos causas

[48] y [49]. Una primera causa considera que la distribución del sistema se debe a las peculiaridades físicas del sistema, mientras que la otra ve la distribución como una forma de resolución de problemas. En el primer caso, la computación distribuida es una necesidad del entorno donde se va a aplicar. Ejemplos de esto son los sistemas de bases de datos distribuidas y los sistemas automáticos de producción o coordinación para toma de decisiones. En el segundo caso, la aplicación distribuida se desarrolla con el fin de conseguir mejor rendimiento que con una versión centralizada de la aplicación. Por ejemplo, obtener en un menor tiempo de ejecución el resultado a un problema concreto, dividiendo el proceso en una serie de tareas o procesos más sencillos que se ejecutan en sitios dispersos comunicados por medio de mensajes.

El funcionamiento de los sistemas distribuidos viene determinado por la utilización de algoritmos distribuidos. Estos algoritmos consisten en un conjunto de procesos que colaboran intercambiando información en la consecución de un objetivo común a todos ellos. Los algoritmos distribuidos resultan complicados desde el punto de vista teórico porque precisan de razonamientos rigurosos que confirmen que el algoritmo cumple las propiedades que se le exigen [50]. En la década de los 90 surgieron métodos formales, tanto para la especificación como la verificación de algoritmos distribuidos que facilitaban su diseño e implementación. Ejemplo de estos métodos son las distintas álgebras de procesos, *CSP* [51] o *CCS* [52], o los modelos basados en estados, tales como el Unity [53] y los autómatas de Entrada/Salida [54].

Las caraterísticas específicas de los sistemas distribuidos impiden en diferentes grados el uso de las técnicas de tratamiento del interbloqueo descritas para sistemas de memoria compartida. Manejar el interbloqueo en un sistema distribuido es más difícil que en un sistema centralizado. Además, operaciones que en un sistema centralizado son sencillas acarrean graves problemas en los entornos distribuidos. La información con la que cuenta cada lugar del sistema acerca del resto de lugares del sistema se consigue mediante la recepción de mensajes. Como la transmisión de mensajes

no es instantánea, la información que incluyen los mensajes puede ser incorrecta a la recepción de los mismos. Así que, la complejidad añadida por la distribución de recursos y procesos, junto con los retardos que se producen en el envío y en la recepción de mensajes, dificultan en gran medida que el tratamiento del interbloqueo se plantee en los mismos términos que en los sistemas de memoria compartida.

Desde el punto de vista práctico, la implantación de algoritmos distribuidos correctos en entornos reales también supone un problema. En muchas ocasiones, las estructuras de datos o la atomicidad de acciones impuestas en la especificación no son factibles en su realización. Estas diferencias entre la especificación real y la teórica favorecen la aparición de fallos en el funcionamiento de los algoritmos. Así mismo, se pueden producir errores una vez implantados los algoritmos, debido a que las restricciones del comportamiento del sistema incluidas en su especificación no se cumplan en el sistema real.

Gran parte de los algoritmos desarrollados para sistemas distribuidos se dieron a conocer en las dos últimas décadas del siglo XX. Se podría hacer incluso una clasificación de ellos teniendo en cuenta si su aparición tuvo lugar antes o después de las revisiones y críticas que realizaron separadamente Knapp [14] y Singhal [55], sobre lo publicado hasta finales de los 80. A partir de estas revisiones la tendencia en la investigación del interbloqueo distribuido cambió radicalmente y se publicaron contraejemplos para algoritmos que se habían dado por válidos. Además, se aceptó la estructura jerárquica de modelos de Knapp [14] y las líneas marcadas por Singhal [55]. Kshemkalyani también sugirió en [56] que las demostraciones de corrección de los algoritmos debían emplear métodos formales que fuesen más rigurosas.

1.3.1. Técnicas básicas para el tratamiento del interbloqueo aplicadas a sistemas distribuidos

El tratamiento del interbloqueo en sistemas distribuidos puede plantearse del mismo modo que se hace en los sistemas centralizados. Sin embargo, la complejidad derivada de la distribución de recursos y procesos, junto con el retardo temporal producido en el envío y en la recepción de mensajes, dificultan que se lleven a cabo los mismos tratamientos.

A continuación, se estudia la utilidad de cada una de las técnicas básicas de tratamiento de interbloqueo en los sistemas distribuidos.

- *Técnicas de prevención.* Estas técnicas no son eficaces porque limitan la concurrencia de los procesos en el sistema. Si el protocolo de ocupación de recursos consiste en que los procesos adquieran todos los recursos que necesitan antes de comenzar la ejecución, la situación de interbloqueo podría darse incluso en la misma fase de adquisición de recursos [14]. En este caso, la condición de retener y esperar no puede evitarse en su totalidad. Este problema podría desaparecer si los procesos adquirieran de uno en uno los recursos, pero esta opción convertiría al sistema en poco eficaz [55]. En muchos casos las técnicas de prevención no son aplicables porque resulta imposible conocer las necesidades de los procesos y no se puede establecer un orden en la asignación de recursos debido a su distribución en el sistema.

- *Técnicas de exclusión.* En estas técnicas se requiere información *a priori* de las necesidades de los procesos. Esto, en algunas aplicaciones, puede que no sea factible porque los recursos necesarios sean desconocidos. Suponiendo que es posible conocer previamente los recursos que necesitan los procesos, habrá que comprobar que el estado global del sistema es seguro tras cada operación de solicitud de recursos. Para ello, todos los lugares deberían pedir, por cada solicitud

de recursos, al resto de lugares su estado para componer el estado global del sistema en cada lugar. Asumir todo este proceso implica que cada lugar tiene que contar con capacidad ilimitada de almacenamiento y de comunicación. Los procesos encargados de chequear si el estado final es seguro tienen que ser mutuamente exclusivos ya que, cuando varios lugares ejecutan concurrentemente estos procesos, pueden llegar a conclusiones erróneas en sus análisis [55].

Resumiendo, con las técnicas de exclusión, a la vez que se necesita conocer de antemano todos los recursos que necesitarán los procesos, se precisa analizar información distribuida por todo el sistema en cada petición de recursos. Por todo ello, estas técnicas resultan difíciles de aplicar y se consideran ineficientes [55].

- *Técnicas de detección.* Las técnicas de detección utilizan algoritmos cuyo objetivo es la búsqueda de circuitos en un grafo. Con estos algoritmos no se limita la concurrencia de los procesos y no se lleva a cabo ninguna acción hasta que el interbloqueo es detectado. Si surge una situación de interbloqueo, ésta se mantiene hasta que es primeramente detectada y, posteriormente, eliminada como efecto de los algoritmos de resolución. En consecuencia, la mayoría de los algoritmos para el tratamiento del interbloqueo en sistemas distribuidos son algoritmos de detección tal y como se pone de manifiesto en [14] y en [55].

En resumen, la utilización de las técnicas de gestión del interbloqueo en sistemas distribuidos provoca diversos problemas. Algunas suponen un coste elevado y otras es imposible aplicarlas. Una idea extendida es, a la vista de la proliferación de algoritmos de detección/resolución, que tanto las técnicas de prevención como las de exclusión son poco recomendables para tratar el interbloqueo en sistemas distribuidos.

1.3.2. Tipos de algoritmos de detección en sistemas distribuidos

El proceso de detección implica dos tareas simultáneas. Por un lado, el algoritmo de detección debe mantener actualizada la información del grafo que representa las interacciones existentes entre procesos y recursos y, por otro lado, debe buscar los circuitos o ciclos que presente el grafo, ya que éste es el síntoma de que hay presente un interbloqueo en el sistema. La búsqueda del circuito depende de la representación de la información del grafo de esperas. Por lo tanto, la clasificación de los algoritmos de detección para sistemas distribuidos se puede establecer de acuerdo a cómo se representa el grafo del sistema y a cómo se realiza la búsqueda de los circuitos. Considerando este último criterio de clasificación, los algoritmos de detección pueden ser de tres tipos: centralizados, jerárquicos o distribuidos.

Los *algoritmos de detección centralizados* disponen de una versión actual del grafo en un único lugar del sistema. Este lugar, denominado lugar de control, puede o bien mantener actualizado el grafo global o bien construir uno nuevo cada vez que realice el proceso de detección, para lo que recopilará el grafo de estado local de todos los lugares del sistema. Desde este lugar también se desarrollará la búsqueda de circuitos. Así mismo, una vez terminada la detección, el lugar de control podrá acometer el proceso de resolución o recuperación. Algunos ejemplos de algoritmos de detección centralizados son: [57] (incorrecto según Gligor), [58], [59] y [60] (incorrecto según Singhal). En [61] se identifican los inconvenientes de este tipo de soluciones centralizadas: una gran vulnerabilidad a los fallos del lugar de control y un tráfico de mensajes elevado para la construcción del grafo global. Las incorrecciones que se señalaron a estos algoritmos tienen su origen en los retrasos de los mensajes que hacen que las informaciones que llegan al lugar de control sean inconsistentes.

El funcionamiento de los *algoritmos de detección jerárquicos* se basa en el orden o

jerarquía establecida entre los lugares del sistema. Los circuitos se hallan descendiendo en esa ordenación tipo árbol de los lugares, esto es, un lugar sólo se encarga de detectar el interbloqueo que pueda producirse en sus lugares-hijo. De esta forma, se consigue descongestionar el tráfico de mensajes con el lugar de control. Algunos ejemplos clásicos de algoritmos de detección jerárquicos son: [62] (incorrecto según Gligor) y [60] (en [63] se presentó un contraejemplo en el que el algoritmo fallaba).

Por último, en los *algoritmos de detección distribuidos*, cada lugar del sistema cuenta con su propio grafo local y se encarga de actualizarlo adecuadamente. Por este motivo, la detección de circuitos cuando están implicados varios lugares debe ser el resultado de la cooperación equitativa de esos lugares. Aunque la búsqueda de interbloqueos no es tan simple como la del caso centralizado, la cantidad de información que se precisa transmitir es inferior porque cada lugar necesita la información del resto de lugares.

Los algoritmos distribuidos son los más difíciles de diseñar y demostrar. Por tanto, no debe sorprender que entre las propuestas publicadas haya soluciones incorrectas en las que se detectan falsos interbloqueos, porque un mismo interbloqueo puede ser detectado simultáneamente en más de un lugar del sistema. La mayoría de los algoritmos de detección desarrollados son distribuidos. Entre los más conocidos se encuentran los de los siguientes investigadores: Goldman [57], Isloor y Marsland [64], Obermarck [65], Chandy [66], Sugihara [67], Mitchell y Merrit [68], Bracha y Toueg [69], Sinha y Natarajan [70] (señalado como incorrecto en [71]), Badal [72], Choudary [71] (demostrado en [56] que es incorrecto), Kshemkalyani [56], González de Mendivil [73] y Kim [74].

A la hora de clasificarlos puede usarse diversidad de criterios. De todos los algoritmos distribuidos de detección y resolución citados anteriormente, se prestará especial atención a los que permitan establecer una comparación de la complejidad adecuada con el algoritmo presentado en esta tesis. Los algoritmos que van a ser de interés en

este trabajo son: [56], [68], [70], [71], [73] y [74]. Todos estos algoritmos de único recurso están basados en la técnica de arco a cazar (*edge-chasing algorithms*). Este tipo de algoritmos usa mensajes especiales o *sondas*. Esos mensajes recorren los caminos que indican las esperas del sistema, pudiendo detectar posibles ciclos de esperas. Una característica que puede diferenciarlos es la capacidad de los nodos para el almacenamiento de la información transmitida por las sondas. En otras palabras, si disponen de información o historia que evite tener que generar las mismas sondas cada vez que el algoritmo se reinicia. La contrapartida de los llamados algoritmos con historia como [56], frente a los que son independientes de la historia como [73], es que tras el aborto de un nodo es necesario actualizar la información no válida que queda en el sistema.

1.3.3. Criterios de corrección

El proceso de detección se puede llevar a cabo ejecutando cualquiera de los tipos de algoritmo presentados. El único requisito que todos ellos deben verificar es que sean correctos. Se dice que un algoritmo de detección es correcto cuando cumple los siguientes criterios [55]:

- *Propiedad de seguridad*. Los algoritmos no deben señalar interbloqueos que no existan. Estos interbloqueos se conocen como *falsos interbloqueos* y son generados por los propios algoritmos de detección.

- *Propiedad de viveza*. Todo interbloqueo debe ser detectado. Los algoritmos tienen que tener la capacidad de localizar todos los interbloqueos que existan en un sistema y además hacerlo en un tiempo finito.

Las principales causas de la detección de falsos interbloqueos son que en el sistema distribuido no hay una memoria global común a todos los lugares y que los lugares

están obligados a comunicarse entre sí por medio de mensajes cuya información representa un estado inconsistente en el grafo, o sea, un estado que no se corresponde con la realidad del sistema.

Al afirmar que un algoritmo cumple la condición de seguridad, se asume que el sistema está libre de abortos espontáneos. Como esta suposición no se puede sustentar en los sistemas reales, decir que un algoritmo satisface la condición de seguridad realmente equivale a decir que la ejecución del propio algoritmo, en ausencia de abortos espontáneos, no genera la detección de falsos interbloqueos.

1.4. Historia reciente del interbloqueo

A finales del siglo XX y en los inicios del siglo XXI, la investigación del problema del interbloqueo ha seguido activa, pero sin la gran proliferación de algoritmos teóricos de las épocas anteriores. En la actualidad la mayor parte de los trabajos que aparecen sobre *deadlock* (interbloqueo) tienen un enfoque práctico.

Todavía siguen apareciendo contribuciones a los campos de: la programación en *ADA* [75], las redes de Petri [76], los sistemas de manufacturación [77] y [78], los sistemas de robots distribuidos [79] y el enrutamiento de paquetes [80] y [81]. También se localizan fácilmente trabajos que presentan dispositivos físicos que realizan la detección del interbloqueo [82] y [83].

Por otro lado, surgen con fuerza trabajos en los que el interbloqueo se relaciona con verificadores de código [84], [85] y [86], o tests de lenguajes de programación como: Java [87] y [88], Go [89], etc. Resulta novedoso el análisis del interbloqueo en Redes de Proceso (*Process Networks*) [90] y la adaptación de técnicas de exclusión de interbloqueos (*avoidance*) para interrupciones temporales en la asignación de recursos (*resource outages*) [91] y para procesamiento de grandes flujos de datos (*stream computing*) [92].

Más íntimamente relacionado con los algoritmos (teóricos) de detección y resolución de interbloqueos, se ha publicado un artículo que resume los fundamentos y el estado del arte del problema del interbloqueo [93].

En lo que respecta a nuevos diseños de algoritmos de detección de interbloqueos se ha encontrado un algoritmo basado en una solución al problema de elección de líder [94]. Ya había sido empleada anteriormente en los algoritmos de Prieto [95] y Castillo [96].

Estos dos algoritmos distribuidos de detección para el modelo único recurso, cumplen los criterios de corrección, anteriormente comentados, y rebajan la complejidad en número de mensajes a costes lineales. La consecución de esta reducción de la complejidad tiene gran importancia porque hasta estos trabajos los algoritmos desarrollados eran cuadráticos y se consideraba imposible llegar a la linealidad. El trabajo de esta tesis avanza en esa línea de investigación. Se muestra que no sólo es posible detectar un interbloqueo formado en $\mathcal{O}(n)$ mensajes, sino que la adaptación del algoritmo a evoluciones dinámicas del sistema mantiene la complejidad lineal.

Capítulo 2

¿Detectar un líder o elegir una víctima?

El propósito de este capítulo es dar respuesta a la pregunta que se formula en su título. Aunque pudiera parecer que se han mezclado los términos de dos conceptos clásicos de la programación distribuida, en la propia cuestión subyace la idea empleada para el diseño del algoritmo de esta tesis: bajo determinadas circunstancias, el problema de detectar un interbloqueo y señalar una víctima que lo resuelva guarda un estrecho parecido con elegir un líder. Ese líder será el que tenga conocimiento de que existe un bloqueo en el sistema.

Una vez se establezca la analogía entre las soluciones de ambos problemas, el capítulo continuará analizando la complejidad de los algoritmos de elección de líder. Se plantea la posibilidad de extender el paralelismo entre ambos tipos de algoritmos a su medida de complejidad. Se fijará como objetivo el diseño de un algoritmo que iguale el coste lineal obtenido para la elección de líder en topologías similares al escenario estático en el que operan los algoritmos de detección y resolución de interbloqueos. Para finalizar, se comentarán algunos de los aspectos de esta primera fase de diseño del algoritmo, que se completará con la incorporación de mecanismos que dinamicen la solución en el capítulo 4.

2.1. Definición del problema de elección de líder

El problema de elección de líder, también llamado de localización de líder, fue expuesto por primera vez por LeLann, quien también propuso su primera solución [97]. El problema plantea cómo conseguir que un sistema, partiendo de una configuración donde todos los procesos poseen el mismo estado, alcance otra configuración donde exactamente uno de los procesos es líder del sistema y el resto han perdido la posibilidad de serlo.

Este tipo de algoritmos es necesario, por ejemplo, si se tiene que ejecutar un algoritmo centralizado y no existe, *a priori*, un proceso señalado que asuma la función de iniciador de ese algoritmo. Este podría ser el caso de un procedimiento de inicialización que debe ser ejecutado en la puesta en marcha de un sistema o después de una caída del sistema [98]. Como el conjunto de procesos activos puede no conocerse previamente no es posible asignar a un proceso el papel de líder para todas las ocasiones.

En los algoritmos que abordan el problema de elección de líder se suele partir de las siguientes suposiciones:

1. Cada proceso del sistema ejecuta una copia del mismo algoritmo local.

2. El algoritmo es descentralizado, es decir, la computación puede ser iniciada por un subconjunto de procesos no vacío y arbitrario.

3. El algoritmo alcanza una configuración final en la que hay exactamente un proceso en estado *líder* y el resto son procesos en estado *perdedor*.

Esta última propiedad a veces se suaviza de manera que el proceso elegido sepa que ha ganado la elección, pero los perdedores no tengan conocimiento de que la han perdido. Nótese que un algoritmo que cumpla esta condición puede verificar fácilmente

la más estricta. Basta con obligar al líder a difundir su estado entre el resto de procesos para que sean informados del resultado de la elección. Esta notificación final se suele omitir en muchos algoritmos de elección de líder. Habitualmente supone una ronda más de mensajes y, por tanto, aumenta la complejidad en número de mensajes.

Muchos algoritmos de elección de líder tienen en común que cada proceso p tiene asociada una variable $state_p$, cuyos posibles valores son *líder* y *perdedor*. A veces se considera un estado previo, *dormido*, en el que el proceso no ha ejecutado ninguna orden del algoritmo, y otro estado, *candidato*, para indicar que el proceso está incluido en la computación, pero aún no es consciente de si es el líder o no. Otros algoritmos también definen estados adicionales para representar distintas situaciones en el proceso de ejecución del algoritmo.

En general, los algoritmos de elección de líder, suponen que el sistema verifica ciertas condiciones [98]:

1. El sistema es totalmente asíncrono. Se asume que los procesos no tienen acceso a un reloj común y que los tiempos de transmisión de los mensajes pueden ser arbitrariamente cortos o largos.

2. Cada proceso se identifica con un nombre único, su identidad, que se conoce inicialmente. Las identidades se eligen de un conjunto totalmente ordenado P, es decir, se dispone de una relación de orden sobre las identidades. La importancia de esta unicidad en el problema de elección de líder es que las identidades puedan ser usadas no sólo para direccionar mensajes, sino también para romper la simetría entre procesos. Cuando se diseña un algoritmo de elección de líder, se podría postular que el proceso con la identidad más pequeña (o alternativamente la mayor) tiene que ganar la elección. En consecuencia, el problema de elección se convierte en encontrar la identidad menor con un algoritmo descentralizado. En este caso el problema se denomina problema de localización de extremos.

Obviamente, un algoritmo que elige el mayor proceso se obtiene invirtiendo las comparaciones entre identidades de un algoritmo que elige el menor proceso.

3. Los procesos se comunican mediante mensajes que pueden contener $\mathcal{O}(w)$ bits, siendo w el número de bits del identificador. Con el fin de comparar convenientemente las complejidades de comunicación de diferentes algoritmos se supone que el tamaño de los mensajes es constante.

2.1.1. Topologías y complejidad

La mayoría de las soluciones al problema de elección de líder que se han propuesto son para redes en anillo, aunque también hay propuestas para topologías más complejas como árboles, cliques o redes completas.

El problema de elección de líder fue tratado en primer lugar para **anillos unidireccionales**. Hay una gran cantidad de trabajos proponiendo soluciones que mejoran la complejidad de mensajes de los algoritmos de elección de líder para esta configuración. La primera solución al problema la aportó LeLann [97]. En su algoritmo, cada iniciador crea una lista con las identidades de todos los iniciadores, después de lo cual se elige el iniciador con la menor identidad. Cada iniciador envía un *token* que contiene su identidad y todos los procesos lo reenvían a través del anillo. Se asume que los canales son *FIFO* y que un iniciador genera su *token* antes de que reciba el *token* de cualquier otro iniciador. Cuando un proceso recibe un *token* ya no puede iniciar el algoritmo. Cuando un iniciador p recibe su propio *token* significa que los *tokens* de todos los iniciadores han pasado por p. Eso implica que p se convierte en el elegido si p cuenta con la menor identidad de las recogidas por el *token*. El algoritmo de LeLann soluciona el problema de elección para anillos usando $\mathcal{O}(N^2)$ mensajes y $\mathcal{O}(N)$ unidades de tiempo. Todos los procesos no iniciadores se consideran perdedores y permanecen infinitamente esperando la recepción de mensajes. La espera puede ser

abortada si el líder envía un token especial por el anillo para anunciar que ha sido elegido líder.

Después de unos años esta solución fue mejorada por Chang y Roberts [99]. Su algoritmo elimina del anillo todos los *tokens* de los procesos que, con toda seguridad, se sabe que van a perder la elección. Para ello, el iniciador p elimina un *token* del anillo si fue creado por un proceso de identidad q tal que $q > p$. El iniciador p se convierte en perdedor cuando recibe un token con identidad $q < p$ y se proclama líder cuando recibe el *token* con su propia identidad p. Este algoritmo conseguía en el caso peor una complejidad $\mathcal{O}(N^2)$, pero una complejidad de $\mathcal{O}(N\log N)$ en un caso promedio cuando todos los procesos son iniciadores. En 1980 Hirschberg y Sinclair propusieron un algoritmo que alcanzaba en el peor caso una complejidad de $\mathcal{O}(N\log N)$ para anillos con canales de comunicación **bidireccionales** [100]. A la vista de ese trabajo, se pensó que la cota inferior para anillos unidireccionales era definitivamente $\mathcal{O}(N^2)$, pero Petersen [101] y Dolev, Klawe y Rodeh [102], de manera independiente, publicaron soluciones de complejidad $\mathcal{O}(N\log N)$. Posteriormente, aparecieron nuevos algoritmos con cotas aún inferiores: Bodlaender [103] encontró una cota inferior de $0{,}34N\log N$ para el peor caso de anillos bidireccionales y Pachl, Korach y Rotem [104] demostraron la existencia de cotas inferiores a $\Omega(N\log N)$ en la complejidad de caso promedio, tanto en anillos unidireccionales como bidireccionales.

Si la topología de red es un **árbol** o se dispone del árbol de expansión de una red, la elección de líder se puede llevar a cabo usando algoritmos diseñados para recorrer estructuras tipo árbol [105]. En esos algoritmos se requiere que, al menos, todas las hojas del árbol sean iniciadoras del algoritmo. Para que el algoritmo progrese en caso de que sólo algunos procesos sean iniciadores, se suele añadir un fase para despertar procesos (*wake-up*). Los procesos que quieren empezar el algoritmo envían un mensaje para despertar a todos los procesos. Una variable booleana permite marcar que todos los procesos envían un mensaje *wake-up* al menos una vez y otra variable cuenta el

número de mensajes de tipo *wake-up* que recibe cada proceso. Cuando un proceso recibe un mensaje *wake-up* a través de cada canal, se inicia el algoritmo para calcular la identidad menor y hacer que cada proceso decida su estado. Cuando un proceso toma esa decisión conoce la identidad del líder. Por tanto, si esta identidad es igual a la del proceso, se convierte en líder y, en caso contrario, se asume que el estado del proceso es el de perdedor.

La elección de líder se resuelve en configuraciones de árbol usando $\mathcal{O}(N)$ mensajes y $\mathcal{O}(D)$ unidades de tiempo. Cuando al menos un proceso inicia el algoritmo, todos los procesos envían mensajes *wake-up* a sus vecinos y cada proceso inicia la ejecución del algoritmo después de recibir el mensaje de cada vecino. Todos los procesos terminan el algoritmo con el mismo valor, la identidad menor de los procesos. El único proceso con esta identidad acabará en estado líder y los otros procesos en estado perdedor. Si los mensajes se pueden reordenar en el canal (el canal no es *FIFO*), los mensaje se tendrán que almacenar temporalmente o ser procesados. Esto puede reducir el número de mensajes, evitando que un iniciador envíe un mensajes *wake-up* a los procesos de los que recibió ya un mensaje *wake-up* y combinando diferentes mensajes que hayan sido enviados por una misma hoja o descendiente del árbol [105].

También se pueden encontrar en la literatura algoritmos para la elección de líder en **redes arbitrarias** de topología desconocida sin conocimiento previo del sistema. Los algoritmos más conocidos llegan a alcanzar una complejidad de mensajes $\mathcal{O}(N\log N)$ y en algunos casos este resultado define una cota inferior $\Omega(N\log N)$. Korach, Kutten y Moran [106] demostraron que hay una estrecha relación entre la elección de líder y los posibles recorridos (*traversal*) para atravesar las redes. Este resultado permite construir un algoritmo de elección eficiente para cualquier clase de redes. La eficiencia de este tipo de algoritmos no tiene en cuenta la complejidad temporal.

La aplicación de los algoritmos de elección de líder en sistemas distribuidos ha propiciado la búsqueda de soluciones para **redes completas**: [107], [108], [109], [110],...

Estas redes están totalmente conectadas y requieren tener previamente un conocimiento global del sistema: número total de nodos, sentido de la dirección, etc. En redes completas los nodos pueden intercambiar mensajes de manera directa y se puede alcanzar un coste lineal tanto en el número de mensajes como en latencia. La propuesta de Villadangos para redes completas [111] sólo precisa la existencia de un anillo virtual que conecte lógicamente los nodos del sistema.

2.2. El interbloqueo visto como una elección de líder dinámica

El problema de la detección de interbloqueo se asemeja en muchos aspectos al de elección de líder en anillos. Se diferencian, fundamentalmente, en que el escenario de la elección de líder es una topología estática en la que se conoce *a priori* el número de nodos del sistema y el sentido de los mensajes. En el caso del interbloqueo se localiza una víctima (líder) en un anillo (ciclo de esperas) que se genera mientras el algoritmo trata de localizar al líder. A pesar de esa diferencia, si se analizan las bases de funcionamiento de diversas soluciones a ambos problemas, se puede ver que son muy similares. En concreto, el funcionamiento de los algoritmos basados en la técnica de *edge-chasing*, como los de González de Mendivil [73] y Kshemkalyani [56], cuando se ponen en marcha en un interbloqueo ya formado, es prácticamente idéntico al de Chang y Roberts [99]. Obviamente, los algoritmos de interbloqueo son más complejos. Han tenido que incorporar mecanismos que permitan recopilar la información de cuáles son los procesos del ciclo mientras se localiza la víctima.

En un algoritmo de detección y resolución de interbloqueos, a diferencia de uno de elección de líder, los nodos son inicialmente activos y sólo se convierten en candidatos a convertirse en víctima, cuando están bloqueados por otro nodo y hay un nodo o

más que esperan por él. Al igual que en el algoritmo de líder, un nodo candidato explora mediante un mensaje (*sonda*) la existencia de otro candidato. Del mismo modo, todos aquellos nodos que al paso del mensaje no eran candidatos se marcan como nodos *dummy* que no intervienen en la detección y resolución del ciclo formado. Si el mensaje recorre el ciclo y vuelve al nodo candidato que lo envió automáticamente se convierte en víctima. Conforme se intercambian mensajes, los nodos candidatos van acumulando información de candidatos con identificadores inferiores que ya no optan a ser los encargados de resolver el interbloqueo. El nodo candidato que recoge las identidades de todos los candidatos del ciclo es el detector del interbloqueo y la futura víctima para su resolución. Es evidente que el estado de los nodos en la configuración final, previa a la resolución del interbloqueo, difiere de la alcanzada en la elección de líder porque en el ciclo pueden coexistir nodos candidatos, nodos *dummy* y un nodo víctima. Los nodos tras la elección de un líder son todos *dummy* ya que, una vez que resulta evidente que no pueden llegar a ser líderes del conjunto, se autodescartan de la competición.

Observando con perspectiva las primeras soluciones para la elección de líder en anillos, se advierte que su complejidad era cuadrática. Los algoritmos de deteccción y resolución de interbloqueo de único recurso basados en la técnica de *edge-chasing* como [70], [56],... comparten con estos algoritmos de elección de líder de la primera época no sólo la forma de detectar/seleccionar víctima/líder mediante el envío de mensajes, sino que alcanzan el mismo nivel de complejidad en número mensajes. Es decir, se ha conseguido recopilar la información de cuáles son los nodos del anillo sin que ello resulte un perjuicio evidente en el número de mensajes necesarios para localizar el líder.

Como se ha descrito en la sección anterior, la complejidad de los algoritmos de elección líder en anillos se fue reduciendo considerablemente. La aparición de soluciones con complejidades no cuadráticas en el problema de líder permite entrever que es

posible la reducción de la complejidad en algoritmos de resolución de interbloqueos.

La propuesta de [111] permite seleccionar un proceso líder en una red completa con tan sólo establecer un anillo virtual que conecte lógicamente a los nodos del sistema. Se puede observar fácilmente que ese anillo virtual guarda un gran parecido con un ciclo de esperas en una situación de interbloqueo. En el anillo lógico, cada nodo conoce el identificador del nodo sucesor y en el ciclo de procesos interbloqueados, donde las esperas entre nodos permanecen fijas hasta la resolución del interbloqueo. Esta información también se puede recopilar durante el proceso de detección.

Aprovechando todas las similitudes, se puede llegar a adaptar a este algoritmo de elección de líder de complejidad lineal, tanto en número de mensajes como en latencia, al problema del interbloqueo. Alcanzar un coste lineal para un algoritmo de detección y resolución de interbloqueo de único recurso es un objetivo conseguido en la tesis doctoral de Prieto [112]. En su algoritmo se desarrolló una adaptación del algoritmo de elección de líder de Villadangos [111] al problema del interbloqueo. Este coste supone un avance notable en el campo de los algoritmos de detección y resolución de interbloqueo. Siguiendo la estela de este algoritmo y utilizando como base la solución de líder para redes completas de Castillo [113], se ha diseñado un nuevo algoritmo para interbloqueo, que mantiene la complejidad lineal y que mejora el coste de [112]. Hay que comentar en ese punto que el algoritmo de elección de líder [113] surgió como corrección a las deficiencias existentes en el comportamiento del algoritmo [111]. La reordenación de los mensajes en distintas configuraciones de nodos afectaban negativamente a la propiedad de seguridad (si un nodo es líder, el resto es *dummy*) y a la de viveza (si se inicia el algoritmo, habrá un líder tarde o temprano) de este último.

El algoritmo que se propone en esta tesis se adapta al proceso de formación y eliminación de esperas, lo que permite que la resolución de interbloqueos, incluso cuando se suceden varios, sea factible sin perder la linealidad. En la mayor parte

de situaciones resultará beneficioso que la ejecución del algoritmo discurra de forma prácticamente paralela a la formación del ciclo de esperas. Esto es debido a que la información útil para resolver el interbloqueo se recopilará progresivamente al igual que se irá eliminando la información obsoleta.

2.3. Algoritmo de elección de líder para redes completas

El algoritmo de elección de líder de [113] asume un sistema de N nodos conectados mediante una red completa, en la que los nodos están organizados en un anillo lógico conocido. Como se observa en la figura 2.1 utiliza sólo cinco variables y precisa de tres tipos de mensajes (*ALG*, *AVS* y *AVSRSP*) para seleccionar como líder del sistema a un iniciador del algoritmo.

Variables: $\forall$ {i, j} $\subseteq \mathcal{N}_n$
  status$_i$: estado del nodo (*passive*, *candidate*, *waiting*, *dummy*, *leader*).
  cand_succ$_i$: identificador del nodo candidato que sucede al nodo i en el anillo virtual.
  cand_pred$_i$: identificador del nodo candidato que precede al nodo i en el anillo virtual.
  neigh$_i$: sucesor del nodo i en el anillo virtual.
  channel(i, j): cola de mensajes que viajan del nodo i al nodo j.

Estado inicial $\equiv$ s$_0$
  $\forall$ n $\in \mathcal{N}_n$: s$_0$.status$_n$ = passive
  $\forall$ n $\in \mathcal{N}_n$: s$_0$.cand_succ$_n$ = s$_0$.cand_pred$_n$ = NULL
  $\forall$ n $\in \mathcal{N}_n$: s$_0$.neigh$_n$: sucesor del nodo n en el anillo virtual.
  $\forall$ (i, j) $\in \mathcal{N}_n \times \mathcal{N}_n$: s$_0$.channel(i, j) = ε

Figura 2.1: Definición de las variables y el estado inicial del algoritmo de elección de líder

En la figura 2.1 se describen estas variables y su estado inicial. Cabe destacar que $neigh_i$ no es propiamente una variable, sino un valor fijado, que proviene del conocimiento previo de la configuración de nodos con la que interacciona el algoritmo

(anillo lógico).

La variable *status* puede tomar cinco valores diferentes. Inicialmente el estado de todos los nodos será *passive*. Sólo los nodos que ejecuten el algoritmo se convertirán en *candidate*. Aquellos candidatos que estén a la espera de conocer el identificador del candidato que les sucede en el anillo virtual pasarán a estado *waiting*. Tras recibir esa información los nodos *waiting* podrán volver ser considerados candidatos a líder o definitivamente se retirarán de la elección adquiriendo el valor *dummy*. El intercambio de mensajes entre candidatos y la comparación de sus identificadores permitirá disminuir el número de candidatos. Cuando sólo quede un nodo candidato en el anillo, se dará por terminada la elección y dicho nodo se convertirá en *leader*.

Durante el proceso de la elección, los candidatos necesitan almacenar, en *cand_succ* y en *cand_pred* respectivamente, los identificadores del candidato sucesor y predecesor que van conociendo. Que esas variables vayan actualizando sus valores es lo que permite que se realicen las comparaciones que determinan qué candidatos deben eliminarse de la elección. Por último, se definen canales de comunicación para el envío de mensajes entre cualquiera dos nodos del sistema.

Respecto a los mensajes que los nodos pueden intercambiar, cada uno tiene una función. El mensaje *ALG* difunde la identidad del nodo candidato que originó el mensaje. La propagación de este mensaje se realiza a través de todos los nodos pasivos que separan al iniciador del mensaje del candidato que le sucede en el anillo lógico. Gracias a este tipo de mensaje todos los candidatos conocen a su candidato predecesor. Cuando un nodo recibe un mensaje que le indica que la identidad de su candidato predecesor es menor que la suya, responde con un mensaje *AVS*. La función de este mensaje es doble. Por un lado, informa al nodo destino de la identidad de su candidato sucesor. Y por otra parte, también le indica que va a dejar de ser candidato y a que nodo, el emisor del mensaje, debe informar de la identidad de su candidato predecesor.

El mensaje *AVSRSP* traspasa la información de un nodo que inició el algoritmo y

```
initiate_i
Pre:status_i = passive.
Eff:status_i ← candidate;
    cand_succ_i ← NULL;
    cand_pred_i ← NULL;
    send[a] ALG(i) to neigh_i.

rcvALG_i(j, init)
Pre:channel(j,i) = ALG(init)·γ.
Eff:channel(j,i) ← γ;
    IF (status_i = passive) THEN
      status_i ← dummy;
      send ALG(init) to neigh_i;
    ELSE IF (status_i = candidate) THEN
      cand_pred_i ← init;
      IF (i = init) THEN
        status_i ← leader;
      ELSE IF (cand_succ_i = NULL) THEN
        IF (i > init) THEN
          status_i ← waiting;
          send AVS(i) to init;
        ELSE (no action)
      ELSE
        send AVSRSP(cand_pred_i) to cand_succ_i;
        status_i ← dummy;
    ELSE (no action).
```

[a]Se denota channel(i, j) ← channel(i, j)·m como 'send m to j' en una acción de i

```
rcvAVS_i(j)
Pre:channel(j,i) = AVS(j)·γ.
Eff:channel(j,i) ← γ;
    IF (status_i = candidate) THEN
      IF (cand_pred_i = NULL) THEN
        cand_succ_i ← j;
      ELSE
        send AVSRSP(cand_pred_i) to j;
        status_i ← dummy;
    ELSE IF (status_i = waiting) THEN
      cand_succ_i ← j;
    ELSE (no action).

rcvAVSRSP_i(j, k)
Pre:channel(j,i) = AVSRSP(k)·γ.
Eff:channel(j,i) ← γ;
    IF (status_i = waiting) THEN
      IF (i = k) THEN
        status_i ← leader;
      ELSE
        cand_pred_i ← k;
        IF (cand_succ_i = NULL) THEN
          IF (k < i) THEN
            send AVS(i) to k;
          ELSE
            status_i ← candidate;
        ELSE
          send AVSRSP(k) to cand_succ_i;
          status_i ← dummy;
    ELSE (no action).
```

Figura 2.2: Algoritmo de elección de líder de redes completas

no tiene posibilidades de ser designado líder del sistema. Un nodo iniciador, en estado *candidate* o *waiting*, no llegará a ser líder si al comparar su identificador con el del candidato que le sucede y el que le precede, se concluye que es inferior a ambos. El destino de un mensaje *AVSRSP* es siempre el candidato sucesor que tenga registrado el nodo que pasa a *dummy* y el contenido es el identificador del predecesor de dicho nodo. Y para terminar, los mensajes *AVS* se encargan de avisar, de forma directa, a los candidatos predecesores de que el identificador recibido mediante un mensaje *ALG* o un mensaje *AVSRSP* es inferior al que posee el emisor del mensaje *AVS*.

En la figura 2.2 aparece el código de las acciones del algoritmo de elección de líder

[113]. Seguidamente, se explica el funcionamiento del algoritmo teniendo en cuenta las precondiciones y los efectos de las mismas.

La elección de líder consta de cuatro fases. En primer lugar se lanza una o varias instancias del algoritmo según el número de nodos que vayan a competir por el liderazgo del sistema (fase de *iniciación*). Esa fase consiste en la ejecución de la acción $initiate_i$. Básicamente los efectos de esta acción permiten distinguir a los nodos que son candidatos a líder de los que no participan. También se inicializan las variables que servirán para recoger la identidad del candidato que precede, $cand\_pred_i$, y sucede, $cand\_succ_i$, al candidato iniciador i. La siguiente fase del algoritmo es la que tiene que ver con la *exploración* parcial del anillo virtual. En esa exploración los candidatos tratan de conocer, mediante el envío de un mensaje ALG, al candidato que le sucede en el anillo. Ese primer mensaje ALG se genera por efecto de la acción $initiate_i$. Después de esta fase, tiene lugar otra en la que el resto de nodos colaboran para hacer llegar el mensaje ALG de un iniciador (fase de *colaboración*). La difusión del mensaje ALG por nodos pasivos hasta alcanzar a otro candidato es consecuencia de la acción $rcvALG_i(j, init)$. Un nodo pasivo se convierte en *dummy* para evitar que lance una nueva instancia del algoritmo, entorpeciendo la elección iniciada.

La siguiente fase del algoritmo es la de *negociación* y surge cuando los mensajes ALG alcanzan un nodo candidato distinto al que lo generó. Esa negociación está supeditada al esquema de prioridades que se quiera implantar. En este algoritmo el esquema implementado establece que el nodo candidato de mayor identificador debe convertirse en líder. En base a esa prioridad, la negociación de los candidatos se realiza comparando los identificadores. En el caso de que el nodo que recibe el mensaje ALG sea el mayor de los dos candidatos implicados, cambiará su estado a *waiting* indicando que, a falta de conocer al candidato que le sucede, sigue teniendo posibilidades de convertirse en líder. Al mismo tiempo, enviará un mensaje directo tipo AVS al nodo candidato que generó ese mensaje. El nodo que recibe un mensaje AVS, si ya dispone

de información de su candidato predecesor en el anillo, se autoexluirá de la elección de líder pasando a estado *dummy*. Antes de convertirse en un nodo *dummy*, ese nodo enviará la identidad de su candidato predecesor en un mensaje *AVSRSP* directo. Sin embargo, si el nodo que recibe el mensaje *AVS* todavía no tiene conocimiento de la identidad de su predecesor, almacenará la identidad del candidato que le sucede (y que es mayor que él) hasta que reciba esta información. En resumen, el proceso de negociación de dos candidatos conlleva la *transferencia* de la información del candidato de menor identificador al mayor y la posterior retirada del candidato menor. La fase de negociación para configuraciones en las que hay más de dos candidatos requiere de nuevas rondas de comparaciones y de intercambio de mensajes *AVS* y *AVSRSP*. En cada una de esas rondas, un candidato dejará de participar en la elección de líder. Finalmente, el candidato mayor recibirá un mensaje *AVSRSP* con información de sí mismo y dará por terminada la elección. El algoritmo también prevé la posibilidad de que haya un único candidato en la red y permite que el mensaje *ALG* de exploración concluya la elección proclamándolo líder.

En resumen, la exploración parcial del anillo virtual por parte de los nodos candidatos y la negociación directa entre candidatos hacen de este algoritmo de elección de líder una solución lineal en número de mensajes y en latencia. Esta característica de coste, junto con la no necesidad de conocer previamente el número de nodos del sistema, son los argumentos que justifican la decisión de trasladar este sencillo algoritmo al funcionamiento de una detección de interbloqueo en un escenario estático. En el capítulo 4 de la memoria, cuando se describa de modo informal el funcionamiento del algoritmo propuesto en esta tesis, quedará patente que la detección de la víctima y la elección de líder se realizan siguiendo las mismas directrices.

2.3.1. Adaptación del algoritmo de líder al algoritmo de detección de interbloqueo en un escenario estático

El diseño del algoritmo de detección de interbloqueo en situaciones estáticas se basa en las cuatro acciones del algoritmo de elección de líder publicadas en [113]. A la hora de comparar ambos algoritmos hay que tener en cuenta que el algoritmo de detección de interbloqueo de esta tesis es mucho más complejo y precisa de una gran cantidad de variables para conseguir su objetivo. Incluye nuevas acciones que amplían su funcionamiento a escenarios dinámicos y las acciones creadas para los casos estáticos se tienen que modificar para incorporar en ellas el comportamiento dinámico. A continuación, se mencionan las semejanzas y diferencias de las acciones necesarias para una detección de interbloqueo estática con las acciones que desempeñan la misma tarea en el algoritmo de elección de líder.

La acción $initiate_i$ del algoritmo de líder (ver figura 2.2) debe tener una réplica homónima en el algoritmo de detección de interbloqueo. Los efectos que se tienen que trasladar de esta acción son: el cambio de estado del nodo que pasa a ser iniciador del algoritmo y el envío de su identificador mediante un mensaje ALG. Hay que advertir que la variable $neigh_i$ debe ser sustituida por una nueva variable en el algoritmo de detección propuesto. Esa nueva variable se definirá como un conjunto y permitirá almacenar las identidades de todos los nodos que esperan por el nodo i (predecesores). En consecuencia, se generarán tantos mensajes ALG como nodos predecesores tenga el nodo i. Cada uno de estos mensajes ALG tendrá como destino a uno de los nodos predecesores del nodo i. La razón de que el mensaje AlG no se dirija hacia el único nodo que le sucede y vaya hacia su/s predecesor/es, estriba en que el algoritmo de detección implementa el modelo SR de petición de recursos, pero un nodo puede recibir muchas esperas. Cambiar el sentido de los mensajes ALG evitará que un mismo nodo esté implicado en un conjunto de computaciones del algoritmo de detección de

interbloqueo.

Los nodos en un estado equiparable al estado *passive* del algoritmo de elección de líder, se comportarán de forma similar ante la recepción de un mensaje ALG (acción $rcvALG_i(j,init)$ de la figura 2.2). Los efectos por los que esos nodos pasan a ser *dummy* y redirigen el mensaje ALG tienen que aparecer también en la detección de un interbloqueo. Debido al cambio de la variable $neigh_i$ ya comentado, en el algoritmo de detección de interbloqueo el reenvío de un mensaje ALG se realizará a todos los predecesores del nodo i.

En lo que respecta al efecto de la acción $rcvALG_i(j,init)$ por el que un nodo *candidate* se convierte en *leader*, el algoritmo de detección de interbloqueo incluirá esta posibilidad intercambiando el estado *leader* por uno que represente la detección de un ciclo. La condición para que se produzca esta situación se asemejará al hecho de que ha circulado un mensaje ALG y ha llegado al nodo que lo originó.

Si el nodo receptor del mensaje ALG es *candidate* se procederá, al igual que la elección de líder, a comparar las identidades de los candidatos implicados. Si el iniciador del mensaje ALG recibido es mayor que la identidad del receptor, se imitará el envío de un mensaje AVS con la misma finalidad. Previo a la comparación, en el algoritmo de la detección de interbloqueo se tendrá que almacenar la identidad del candidato emisor del mensaje ALG. Es obvio que, a causa del sentido de los mensajes ALG en la detección de interbloqueo, ese candidato iniciador será considerado un sucesor en vez de un predecesor como ocurre en la elección de líder.

Al generarse un mensaje AVS, en el caso de la solución de líder, surge un estado intermedio *waiting*. Ese estado representa que el candidato receptor del mensaje ALG sigue en la carrera para convertirse en *leader* y le falta conocer si el candidato que le sucede es mayor que él. El estado *waiting* de la elección de líder no tendrá equivalencia directa en un estado en el algoritmo de detección de interbloqueo. Sin embargo, cuando un nodo mantiene su estado *candidate* tras recibir un mensaje ALG en la elección

de líder, queda patente que no tiene posibilidades de ser líder y se desvincula de la elección si ya conoce la identidad de su predecesor. Es importante explicar en este punto que un candidato que abandona el proceso de elección se convierte en *dummy*, enviando un mensaje *AVSRSP*. Por el contrario, en la detección de interbloqueo que se propone, el candidato que se salva de ser señalado víctima tendrá que permanecer en estado *candidate* para atender a la dinámica del sistema. En evoluciones posteriores del sistema este candidato puede participar en una nueva detección ahorrándose y ahorrando al sistema recoger de nuevo información que no ha variado, o sea, lanzar de nuevo una instancia del algoritmo.

Por último, las acciones $rcvAVS_i(j)$ y $rcvAVSRSP_i(j,k)$ del algoritmo de líder descritas en la figura 2.2 también tendrán que aparecer en el algoritmo de detección de interbloqueo. Como puede observarse la acción $rcvAVSRSP_i(j,k)$ dispone de un efecto por el que un nodo en estado *waiting* pasa a ser *leader* al recibir un mensaje *AVSRSP*. La información que transporta ese tipo de mensaje hace alusión al identificador de candidato predecesor vigente que hasta el momento de la recepción desconocía. Dado que los mensajes *AVS* y *AVSRSP* recorren el anillo lógico en sentidos opuestos y que el algoritmo de detección de interbloqueo debe explorar el ciclo de esperas buscando a sus candidatos predecesores, se presupone que el efecto señalar víctima tendrá que producirse en la acción que se encargue del tratamiento de los mensajes *AVS*.

Razonando de forma similar, se puede llegar a la conclusión de que, si la acción $rcvAVS_i(j)$ del algoritmo de elección de líder es la encargada de actualizar el valor de la variable $cand\_succ_i$, en el algoritmo de detección de interbloqueo esa función la tendrá que desempeñar la acción que procese los mensaje *AVSRSP*. Del mismo modo, la actualización de la variable $cand\_pred_i$, que en la elección de líder la lleva a cabo la acción $rcvAVSRSP_i(j,k)$, deberá ser realizada por la acción que en la detección de interbloqueo gestiona los mensajes *AVS*.

Si se repasan las acciones $rcvAVS_i(j)$ y $rcvAVSRSP_i(j,k)$ de la elección de líder

considerada (figura 2.2), se puede observar que los candidatos, que abandonan la elección y pasan a ser *dummy*, verifican una condición: conocen tanto a su candidato sucesor como a su candidato predecesor y saben que su identificador es el menor de los tres. Esta idea también se empleará en el algoritmo de detección de interbloqueo y será el fundamento del envío del mensaje de un nodo que no puede ser víctima, porque asumirá que su identificador es inferior al de los candidatos con los que ha intercambiado información.

Capítulo 3

Especificación del problema

Antes de describir en detalle el algoritmo que se ha diseñado, es necesario especificar el problema al que se pretende dar solución. Con una idea clara de cuál es el problema que se aborda y qué requisitos debe cumplir la solución que se adopte, se podrá determinar si el algoritmo es una solución correcta. Del mismo modo, gracias a la especificación del problema, también se podrá dilucidar si un sistema cumple los requisitos necesarios para emplear las soluciones encontradas.

Por todo ello, el principal objetivo de este capítulo es proporcionar la especificación del problema que se va a tratar. En primer lugar se va a exponer de una manera algo informal el modelo de sistema en el que el algoritmo debe funcionar. Seguidamente, se describirá formalmente el sistema mediante el uso de un autómata de Entrada/Salida. La definición del autómata, designado abreviadamente como G, va a permitir modelar el comportamiento del sistema en el que se desea trabajar. El autómata G se ha construido pensando en una dinámica concreta del medio, pero es muy poco restrictivo por lo que puede modelar igualmente el comportamiento de otros sistemas más restrictivos.

La parte final del capítulo está dedicada a la especificación formal del problema, E, que consiste básicamente en imponer al autómata G los criterios de corrección exigidos a la solución (algoritmo). De todas las posibles ejecuciones del autómata G

sólo aquellas que verifiquen las propiedades de seguridad y viveza se considerarán comportamientos válidos e integrados en *E*.

Las soluciones que se han ido proponiendo al problema del interbloqueo a lo largo del tiempo son algoritmos específicos para un modelo de petición de recursos. El tipo de de peticiones que admite el sistema viene determinado básicamente por la aplicación que desarrollen sus procesos. Aunque sería interesante contar con un algoritmo que resolviera el problema del interbloqueo para cualquier modelo de petición de recursos, se prefiere diseñar algoritmos para modelos de asignación concretos en sistemas concretos buscando maximizar su eficacia. El algoritmo ideado se ha creado para aplicaciones de único recurso.

En el presente capítulo se va a presentar de manera formal el gestor de procesos y recursos que gobierna el sistema y sigue el modelo de petición de único recurso. Para ello, se hará uso de una herramienta, que también servirá para la demostración de corrección del algoritmo distribuido. La herramienta elegida se base en el modelo de Autómatas de Entrada/Salida introducido por Lynch y Tuttel en [54] y [114] respectivamente. Este modelo va a permitir diferenciar entre acciones controladas por el algoritmo y acciones que controla el medio. Como las acciones del medio pueden tener lugar en cualquier situación, restringir el comportamiento del entorno para modelarlo es una tarea bastante complicada en el proceso de formalización. Sin embargo, superada esa tarea, la representación general del gestor permite rebajar la dificultad del diseño del algoritmo de resolución de interbloqueos ya que el algoritmo no tiene que considerar las operaciones propias del gestor del medio. Además, la descripción formal del gestor tratará de reflejar el estado de las esperas desde el punto de vista local. Esta visión favorecerá la comprensión de las transiciones entre estados posibles del sistema.

La descripción del funcionamiento del entorno mediante un autómata de Entrada/Salida se complementará con la representación de las esperas del sistema en un

grafo. Los efectos de las acciones del gestor del sistema modificarán dicho grafo. La combinación del autómata y de este grafo da lugar a un modelo completo para un sistema *SR* en el que no hay abortos espontáneos de procesos. El modelo del gestor favorecerá la caracterización formal de una situación de interbloqueo en el sistema.

3.1. Modelo del sistema

Uno de los motivos por los que se puede encontrar algoritmos incorrectos publicados es la falta de formalismo utilizada en su diseño [56]. Si se incluye dicho formalismo, se facilita notablemente la demostración de corrección de los algoritmos. La mayor parte de los trabajos publicados en la actualidad presentan los algoritmos junto con sus pruebas de corrección, pero rara vez se da una descripción formal del sistema con el que interactúa el algoritmo. Formalizar el funcionamiento del entorno de ejecución del algoritmo, aunque no lo parezca a simple vista, ayuda a delimitar los comportamientos del sistema y a enunciarlos como propiedades.

3.1.1. Descripción informal del sistema

La descripción que se ofrece en esta sección tiene como objetivo dar una visión sencilla de las componentes y del funcionamiento del sistema. Los sistemas para los que se diseña el algoritmo de esta tesis son distribuidos. Este tipo de sistemas está formado por un conjunto de lugares interconectados mediante una red de comunicaciones. Cada uno de estos lugares cuenta con una serie de recursos propios (locales) pero los procesos que se ejecutan en cada uno de los lugares pueden requerir no sólo recursos locales, sino también recursos de otros sitios (remotos). El acceso a los recursos locales, la ejecución de los procesos de una misma ubicación y la comunicación entre procesos de ubicaciones diferentes se realiza bajo la supervisión de un gestor local. La combinación de los gestores locales de los distintos lugares del sistema constituye, lo

que se denomina habitualmente, el gestor del sistema. Esto evidencia que el gestor de procesos y recursos de un sistema distribuido ha de ser distribuido. Sin riesgo a perder la generalidad de la definición dada de sistema distribuido, se puede asimilar un lugar a un único proceso o recurso. Esta simplificación, por tanto, permite considerar el sistema como un conjunto de nodos cuando no es necesario distinguir entre procesos y recursos. En consecuencia, en la mayor parte de este trabajo, se establecerá que el sistema está formado por un conjunto numerable de nodos, $\mathcal{N}$, identificados de manera única y en él hay establecida una relación de orden total $(\mathcal{N}, \prec)$.

Otro elemento del sistema que hay que describir es la red de comunicaciones. Como los nodos del sistema deben intercambiar información mediante mensajes es importante indicar las características de los canales por los que se transmiten dichos mensajes. Se supondrá que existe un canal de comunicaciones lógico conectando todo par de nodos del sistema. Por otro lado, el funcionamiento de los canales es ideal, esto es, los canales son fiables porque no se producen ni pérdidas ni duplicidades de mensajes y, aunque pueden demorarse, los mensajes no sufren retardos infinitos. Asumiendo estas propiedades, se puede asegurar que un nodo que ha mandado un mensaje sabe con certeza que será recibido, pero no sabe exactamente cuándo llegará. En todo caso puede saber que ha sido recibido si el nodo receptor le manda un mensaje de confirmación.

Si se considerara un sistema de comunicaciones más realista, habría que tener en cuenta la pérdida de mensajes y deberían incorporarse en el sistema mecanismos de repetición de mensajes y métodos de ordenación para descartar mensajes repetidos o, simplemente, mensajes que ya no son válidos. Resulta obvio que incluir todos estos aspectos, relacionados con la ordenación correcta de mensajes, incrementa la complejidad de las acciones del sistema gestor. Para evitar que esa complejidad se extienda a la especificación del problema y al algoritmo que lo va a resolver y, sabiendo que el esquema de asignación y liberación de recursos por parte de un proceso no se ve

modificado al hacer que los canales de comunicación tengan un comportamiento más cercano a la realidad, se opta por trabajar con una red de comunicaciones idealizada.

El modelo de petición de recursos que se admite en el sistema es el de único recurso. Como ya se comentó en el capítulo de introducción, este tipo de petición consiste en que los procesos solicitan los recursos que necesitan uno a uno y los recursos son de exclusión mutua, sin la posibilidad de que haya más de un proceso utilizando el mismo recurso simultáneamente. Cuando un proceso solicita el acceso a un recurso, el proceso queda bloqueado y permanece a la espera hasta que le es concedido el uso del recurso. Una vez que el recurso pueda utilizarse, el estado del proceso vuelve a ser activo. En este estado el proceso puede solicitar el acceso a un nuevo recurso o liberar alguno de los recursos ya asignados. Mientras el recurso está asignado a un proceso, ningún otro proceso puede acceder a él. Sólo cuando el recurso sea liberado por el proceso que lo tiene asignado, podrá ser utilizado previa solicitud.

Considerando el sistema como un conjunto de nodos, el modelo de petición de único recurso se traduce en que un nodo espera como máximo por otro nodo y únicamente los nodos activos envían mensajes. En consecuencia, un nodo activo puede bloquearse en cualquier instante de tiempo y volverse a activar cuando el nodo por el que espera lo permita.

Aunque se supone que el número de tareas que realiza un proceso en su ejecución es finito, la ejecución concurrente de varios procesos puede conducir a una situación de interbloqueo del sistema. Si en el sistema sólo hubiera un proceso, es obvio que no podría surgir este problema y, transcurrido el tiempo en el que se desarrollan todas las tareas, los recursos quedarían libres y listos para otro uso. Al tratarse de un sistema distribuido, cada uno de los nodos que lo conforman debe ejecutar una copia del algoritmo de detección y resolución de interbloqueo que se ha diseñado. La resolución de un posible interbloqueo implica el aborto de un nodo que representa a un proceso. Para ello, se cancelan las solicitudes de recursos que no han sido todavía

atendidas, se liberan los recursos a los que tenía acceso y se restaura su estado inicial a todos los recursos liberados [14]. En el sistema se presupone que no se dan abortos espontáneos, sino que se requiere un proceso de detección, y que nunca abortarán nodos que no estén interbloqueados.

El comportamiento de los nodos vendrá dado por la información que guardan de su entorno, las relaciones de espera que existen en el sistema, y la información que se obtiene de la ejecución del algoritmo de detección y resolución de interbloqueos.

3.1.2. Descripción formal del gestor del sistema

3.1.2.1. Grafo de Esperas

Para modelar el estado de las esperas en sistemas distribuidos se suele utilizar un grafo ya que esta representación gráfica es sencilla, pero aporta un nivel de abstracción suficiente para que la solución del problema del interbloqueo sea lo más general posible.

Los grafos más frecuentemente usados para representar las esperas de un sistema concurrente son el *Grafo de Asignación de Recursos* (*Resource Allocation Graph - RAG*) y *Grafo de Esperas* (*Wait-For Graph - WFG*) [14] [55]. Estos grafos no son adecuados para modelar el estado de las esperas en sistemas distribuidos porque las acciones de petición, asignación y liberación de recursos implican mensajes entre distintos lugares del sistema. Los mensajes que surgen en estas acciones pueden reordenarse ya que los mensajes requieren un tiempo de transmisión. Estos retardos hacen que la información de las esperas del sistema que se desprende del grafo no sea real en todos los instantes de tiempo [56] [115].

El grafo de esperas que se va a adoptar es un 1-grafo bipartito, sin lazos, dirigido y dinámico, $G \equiv (\mathcal{N}, \mathcal{A})$, donde $\mathcal{N}$ es el conjunto de nodos del sistema y $\mathcal{A}$, el conjunto de arcos que representan las relaciones entre los nodos [116]. Se dice que es bipartito

porque tiene dos elementos constitutivos, nodos y arcos. En el grafo no aparecen lazos porque no hay arcos cuyo extremo inicial y final coincidan. La multiplicidad de los arcos es como máximo la unidad, es decir, un arco aparece a lo sumo una vez. El grafo es dirigido porque los arcos se caracterizan por su extremo inicial y final. Como el conjunto de arcos, $\mathcal{A}$, varía con el tiempo, se trata con un grafo dinámico.

El conjunto de nodos $\mathcal{N}$ representan elementos reales de la computación (procesos o recursos). Los arcos de este tipo de grafo pueden unir dos nodos reales o, debido a la dinámica del medio, apuntar a un nodo sin estar unido a un nodo inicial o viceversa, surgir de un nodo inicial sin apuntar al nodo final. Aunque propiamente no lo son, estos dos últimos tipos de enlaces van a ser denominados también arcos porque sirven para modelar la asincronía en el conocimiento de las modificaciones de las esperas. La existencia de un arco que une el nodo inicial de la espera, nodo i y se dirige a otro nodo j sin ser apuntado implica que el nodo i está esperando por el nodo j aunque el nodo j desconoce que lo espera en ese instante. Por otro lado, el arco que va desde un punto intermedio del arco que uniría al nodo i con el nodo final de la espera, nodo j, conlleva que el nodo j tiene información según la cual está siendo esperado por el nodo i, pero éste ha dejado de esperarle.

3.1.2.2. El autómata del gestor del sistema, *G*

Para la descripción formal del funcionamiento del sistema se va a utilizar un autómata de Entrada/Salida. Este modelo matemático permite representar el sistema de forma abstracta haciendo uso de una máquina de transiciones de estado. En la definición de un autómata de Entrada/Salida se incluye un conjunto de estados, $states(A)$, un conjunto de estados iniciales, $start(A)$, una signatura de acciones, $sig(A)$, una relación de transición que define los pasos del autómata o acciones que pueden ser ejecutadas en cada estado, $steps(A)$ y una relación de equivalencia que especifica las ejecuciones equitativas del autómata, $part(A)$.

Estado de G:

El estado de un autómata describe con precisión en que situación se encuentra el sistema y viene determinado por el valor de un conjunto de variables. El autómata del medio, G, modela un sistema distribuido formado por un conjunto de nodos $\mathcal{N}$ y los canales de comunicación que unen cada par de nodos. Los nodos del sistema tendrán un número único como identificador tomado del conjunto de los números naturales por lo que se puede afirmar que en el conjunto de nodos del sistema, $\mathcal{N}$, hay una relación de orden definida. Como los nodos del grafo de esperas representan tanto procesos como recursos por nodos, se asumirá que los identificadores que representan a los recursos son inferiores que los que corresponden a los procesos.

Las variables que definen el estado del autómata del medio, G, se muestran en la figura 3.1. En ella se puede observar un nodo i viene caracterizado por las variables t_activ_i, $blocker_i$, $set\_waiters_i$ y $state_i$.

```
Estado (s) del autómata del medio G, s ∈ states(G):
 ∀ i ∈ 𝒩: state_i ∈ {active, blocked, aborted} //Estado del nodo i
 ∀ i ∈ 𝒩: t_activ_i ∈ ℕ //Tiempo de activación del nodo i
 ∀ i ∈ 𝒩: blocker_i ∈ 𝒩 ∪ NULL //Nodo por el que el nodo i está esperando
 ∀ i ∈ 𝒩: set_waiters_i ⊆ {(n, t, b) con n ∈ 𝒩, t ∈ ℕ ∧ b ∈ {sent, received, released}}
    //Conjunto de tuplas (nodo, tiempo, estado espera) de los nodos que esperan por el nodo i
```

Figura 3.1: Definición del estado del autómata del medio, G.

La variable $\boldsymbol{t\_activ_i}$ permitirá registrar el instante de tiempo en el que el nodo i se desbloquea. Durante el periodo en el que el nodo permanezca bloqueado, su tiempo de activación quedará fijado al del instante en el que pasó de estado bloqueado a activo por última vez. Inicialmente, todos los nodos del sistema son activos y presentan el mismo tiempo de activación.

Las variables $blocker_i$ y $set\_waiters_i$ representan la información de la que dispone el nodo i sobre sus relaciones de espera con otros nodos.

En $\boldsymbol{blocker}_i$ se almacena el identificador del nodo por el que el nodo i está esperando. En caso de que el nodo no esté esperando por ningún otro nodo en $blocker_i$ aparece el valor *NULL*. Como el problema de interbloqueo que se aborda en este trabajo trata el interbloqueo para sistemas con modelo de asignación de único recurso, $blocker_i$ guarda la identidad de un único nodo como máximo. Si el nodo i es un proceso, $blocker_i$ contendrá el nombre del recurso que ha solicitado y, por el contrario, si es un recurso, $blocker_i$ incluirá la identidad del proceso al que está asignado.

La variable $\boldsymbol{set\_waiters}_i$ recoge información del conjunto de nodos que están esperando por el nodo i. Los elementos de $set\_waiters_i$ son ternas en las que, además de la identidad del nodo que está esperando al nodo i, aparece el tiempo de activación de ese nodo en el momento en el que surgió esa espera y el estado de esa espera. Los tres valores posibles para definir el estado de una espera son: *received*, *sent* y *released*. En un sistema distribuido, puede suceder que el nodo i tenga pleno conocimiento de las esperas que le atañen (*received*), que las esperas todavía no se hayan consolidado (*sent*) o que las esperas ya no sean completas porque se haya iniciado el borrado de las mismas (*released*). En este último caso, los nodos de $set\_waiters_i$ seguirán bloqueados por el nodo i porque aún no han sido notificados convenientemente de la nueva situación. A pesar de ello, el nodo i ya se habrá liberado de la relación con los procesos que lo requerían como recurso o con los recursos que le habían sido asignados siendo proceso.

Por otra parte, la variable $\boldsymbol{state}_i$ denota el estado del nodo i. Los posibles valores que puede adoptar esta variable son *active*, *blocked* o *aborted*. Un nodo activo no espera por ningún otro nodo. Si el nodo es un proceso, éste podrá solicitar el acceso a los recursos del sistema. En cambio, si el nodo representa a un recurso, al estar activo podrá ser asignado a un proceso que esté esperando por él. Cuando un nodo está bloqueado, el nodo está esperando por algún otro nodo. En caso de que el nodo sea un proceso, el nodo espera que el recurso que ha solicitado previamente le

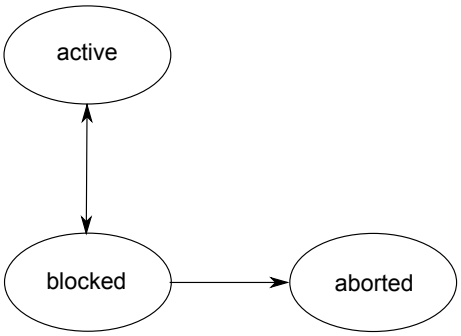

Figura 3.2: Transiciones del valor de la variable *state* en G.

sea concedido. Para un nodo de tipo recurso, estar bloqueado significa que el recurso espera ser liberado por el proceso al que está asignado. Por último, un nodo abortado ni espera ni es esperado por otro nodo. Si un proceso aborta, permanecerá indefinidamente en ese estado. Si el proceso tiene que volverse a ejecutar, aparecerá en el sistema representado como un nuevo nodo.

La variable $state_i$ puede ver modificado su valor en una ejecución de G, pero cualquier modificación no es válida. La figura 3.2 muestra los cambios de estado posibles. Un nodo activo puede bloquearse en cualquier momento y al desbloquearse el nodo volverá ser activo de nuevo. Si el algoritmo de detección y resolución de interbloqueo hace que un nodo interbloqueado se tenga que abortar, el estado del nodo pasará de bloqueado a abortado directamente.

La definición del estado inicial del autómata permite conocer con precisión las condiciones de partida en la puesta en marcha del autómata. Los valores de las variables del autómata G en el momento de iniciarse el funcionamiento del sistema determinarán su comportamiento. El estado inicial de G queda fijado formalmente en la figura 3.3.

Se considerará que el sistema parte del reposo, o sea, que los recursos están libres, ningún proceso espera o retiene recursos y todavía no ha habido intercambio de mensajes entre los nodos del sistema. En consecuencia, en el estado inicial s_0 todos

Estado inicial (s_0) del autómata del medio G:
$\forall$ i $\in \mathcal{N}$: $s_0.state_i$ = active
$\forall$ i $\in \mathcal{N}$: $s_0.t\_activ_i = 1$
$\forall$ i $\in \mathcal{N}$: $s_0.blocker_i$ = NULL
$\forall$ i $\in \mathcal{N}$: $s_0.set\_waiters_i = \emptyset$

Figura 3.3: Estado inicial del autómata del medio, G.

los nodos están activos, no hay esperas entre los nodos ni mensajes en los canales de comunicación pues quedarían reflejadas en *set_waiters*

Signatura de G:

La signatura de acciones del autómata del medio describe el interfaz entre el autómata y su entorno. Las acciones se clasifican en: acciones de entrada, acciones de salida y acciones internas. Se denominan acciones de entradas a aquellas que no controla el autómata porque siempre están habilitadas y se pueden ejecutar en cualquier instante de tiempo. A diferencia de las acciones de entrada, las acciones de salida y las acciones internas son controladas por el autómata, es decir, su ejecución queda habilitada en función del estado alcanzado por el autómata.

De las acciones del autómata del medio, las de más interés son las que tienen que ver con las esperas del sistema. Por el contrario, las acciones que no afectan a las variables de estado ya definidas, se considerarán acciones internas del autómata y no aportarán información.

En la figura 3.4 se expresa formalmente la signatura de las acciones del autómata G. Como puede observarse, no existen acciones internas. Las acciones de salida comprenden tanto a las acciones de formación de las esperas, $StartAddArc_i(j)$ y $EndAddArc_i(j,\ t)$, como a las de borrado, esto es, $StartDelArc_i(j,\ t)$ y $EndDelArc_i(j)$. Además, también se incluye una acción de entrada $Abort_i$. Esta acción indica cómo

$$\begin{aligned}
&\texttt{Input}(G) = \{\texttt{Abort}_i \text{ tal que } i \subseteq \mathcal{N}\} \\
&\texttt{Internal}(G) = \emptyset \\
&\texttt{Output}(G) = \{\texttt{StartAddArc}_i(j),\ \texttt{EndAddArc}_i(j,t),\ \texttt{StartDelArc}_i(j,t),\ \texttt{EndDelArc}_i(j) \\
&\qquad\qquad \text{tal que } \{i,\ j\} \subseteq \mathcal{N} \wedge i \neq j \wedge t \in \mathbb{N}\}
\end{aligned}$$

Figura 3.4: Signatura de las acciones del autómata del medio, G.

debe resolverse un interbloqueo cuando el autómata recibe información de su existencia.

Acciones de G:

En este apartado se van a definir formalmente las acciones del autómata G y se van a describir sus posibles transiciones. Las acciones de un autómata determinan el comportamiento del sistema ya que definen las transiciones entre posibles estados del sistema.

En la figura 3.5 se muestran las acciones del autómata del medio siguiendo el esquema de precondición-efecto. Una acción π queda habilitada en un estado s si, y sólo si, satisface su precondición. Esta precondición acota los posibles estados del sistema en los que la acción se puede ejecutar, permitiendo la transición a otros estados del sistema. La ejecución de la acción hace que el sistema alcance un nuevo estado $s\prime$. En esta transición de estado se modifican las variables tal y como indican los efectos de la acción.

Es importante señalar que la ejecución de las acciones se realiza de forma atómica impidiendo que se produzcan incongruencias en las asignaciones de valores de las variables. La asignación de nuevos valores se escribe de tal forma que la aparición de la variable a la izquierda de la orden representa el valor de la variable tras la ejecución de la acción. Cuando el nombre de una variable aparece en cualquier otra localización, en las precondiciones o en los efectos de las acciones, la variable hace referencia a su valor anterior a la ejecución de la acción. De esta manera, no es necesario indicar

```
StartAddArc_i(j)
precondiciones:  state_i = active.
efectos:
  state_i:= blocked;
  blocker_i:= j;
  set_waiters_j:= set_waiters_j ∪ {(i, t_activ_i, sent)}.

EndAddArc_i(j,t)
precondiciones:  state_i ≠ aborted ∧ (j, t, sent) ∈ set_waiters_i.
efectos:
  set_waiters_i:= set_waiters_i ∪ {(j, t, received)} \ {(j, t, sent)}.

StartDelArc_i(j,t)
precondiciones:  (j, t, received) ∈ set_waiters_i ∧ (state_i = active ∨ state_j = aborted).
efectos:
  set_waiters_i:= set_waiters_i \ {(j, t, received)};
  IF (state_i = active) THEN
    set_waiters_i:= set_waiters_i ∪ {(j, t, released)}
  ENDIF.

EndDelArc_i(j)
precondiciones:  blocker_i = j ∧ ((i, t_activ_i, released) ∈ set_waiters_j ∨ state_j = aborted).
efectos:
  IF (state_j ≠ aborted) THEN
    set_waiters_j:= set_waiters_j \ {(i, t_activ_i, released)}
  ELSEIF ((i, t_activ_i, sent) ∈ set_waiters_j) THEN
    set_waiters_j:= set_waiters_j \ {(i, t_activ_i, sent)}
  ENDIF;
  state_i:= active;
  blocker_i:= NULL;
  t_activ_i:= t_activ_i + 1.

Abort_i
precondiciones:
efectos:
  state_i:= aborted;
  blocker_i:= NULL;
  t_activ_i:= t_activ_i + 1;
  set_waiters_i:= ∅.
```

Figura 3.5: Acciones del autómata del medio, G.

el estado para cada una de las apariciones de una variable en el autómata. Después de estas aclaraciones sobre el modelo y la notación empleada para la escritura del autómata, se va a proceder a explicar en detalle cada una de las transiciones del autómata G.

Acción $StartAddArc_i(j)$:

La precondición de esta acción exige que sólo la ejecuten los nodos que se mantienen activos (nodo i en la figura 3.6.a). La transición indica que el nodo i empieza a esperar al nodo j.

Los efectos de la acción consisten en un cambio de estado del nodo i y la asignación de la identidad del nodo j a la variable $blocker_i$. Es obvio que, tras ejecutarse esta acción, el nodo i deja de ser un nodo activo y pasa a estar bloqueado, con lo cual el nuevo estado del nodo i será *blocked*. Además, se registrará el nodo por el que i queda bloqueado y se anotará que el nodo i ha iniciado una espera por j con un determinado tiempo de activación. Esa información, (i, t_activ_i, *sent*), se incorporará a $set\_waiters_j$ (figura 3.6.b). Nótese que en esta acción no hay ninguna restricción que impida que un nodo se bloquee por un nodo abortado.

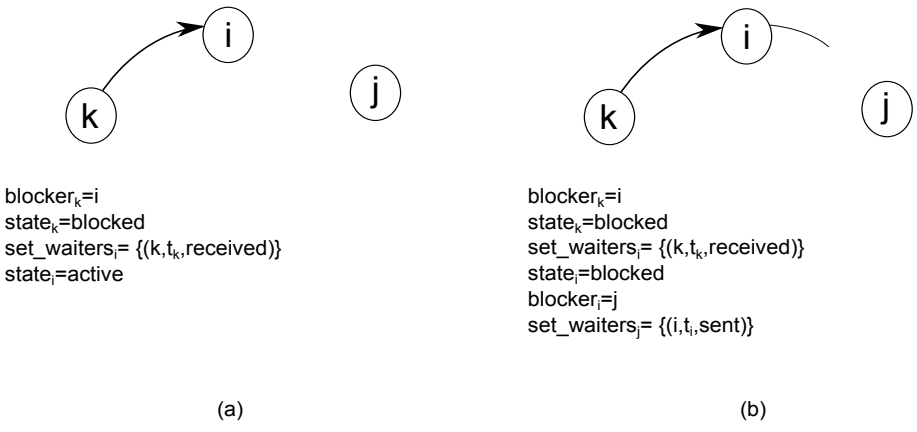

Figura 3.6: Precondiciones (a) y efectos (b) de la acción $StartAddArc_i(j)$.

Acción *EndAddArc*$_i$(j, t):

Esta acción permite completar una espera iniciada por el nodo j hacia el nodo i siempre que el estado del nodo bloqueante i no sea *aborted*. Con la exclusión del estado *aborted* del nodo i, se logra que un nodo abortado no vuelva a participar activamente en la evolución posterior del sistema. El registro de la espera antes de la ejecución *EndAddArc*$_i$(j, t) está incluido en *set_waiters*$_i$ como (j, t, *sent*), donde t es el instante de tiempo en el que el nodo j era aún activo, es decir, antes de que se bloqueara por el nodo i (figura 3.7.a). El único efecto de la acción consiste en sustituir los datos de esa espera, la cual parte del nodo j, por los datos de la espera ya completada. Así que, la variable *set_waiters*$_i$ contendrá la tupla (j, t, *received*) en lugar de (j, t, *sent*) cuando se ejecute la acción (figura 3.7.b).

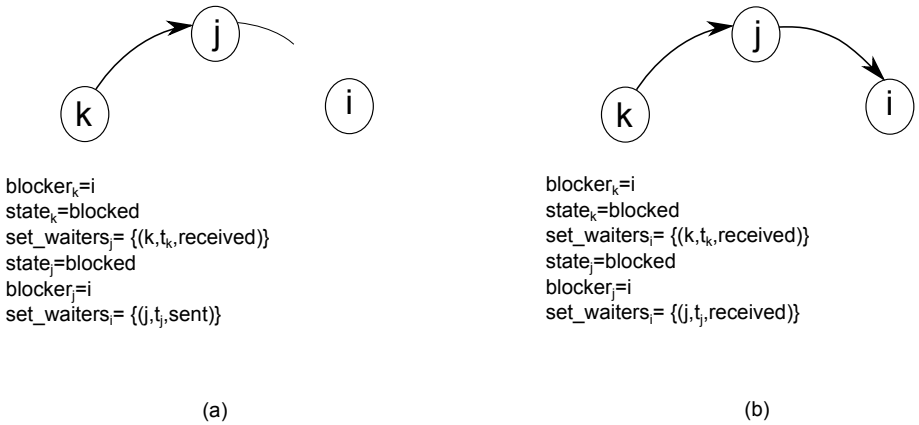

Figura 3.7: Precondiciones (a) y efectos (b) de la acción *EndAddArc*$_i$(j, t).

Acción *StartDelArc*$_i$(j, t):

Con esta acción, o bien se inicia el proceso de activación del nodo j (figuras 3.8.a y 3.8.b), o bien se elimina la espera residual entre el nodo j y el nodo i que permanece en el sistema tras el aborto del nodo j (figura 3.8.c y 3.8.d). En la primera situación, el nodo i queda desligado del bloqueo que el nodo j mantenía por él. En la segunda situación, los efectos de la acción *Abort*$_j$ han suprimido el valor de la variable *blocker*$_j$

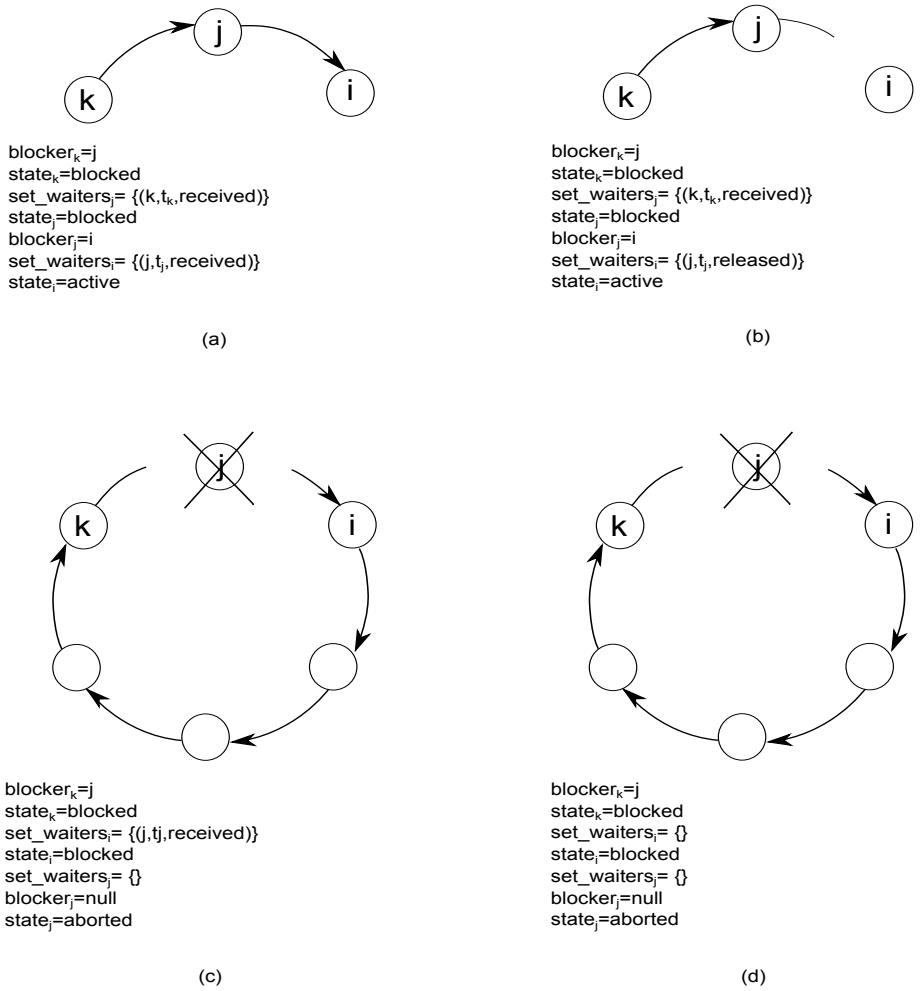

Figura 3.8: Precondiciones (a)-(c) y efectos (b)-(d) de la acción $StartDelArc_i(j,\ t)$.

rompiendo definitivamente el bloqueo por el nodo i. Que el nodo j considere que ha dejado de esperar por el nodo i se debe a que los efectos de la acción $Abort_j$, además de transformar el estado del nodo j en *aborted*, han anulado el valor de la variable $blocker_j$.

La precondición para la ejecución de la acción $StartDelArc_i(j,\ t)$ recoge las dos transiciones descritas previamente. La condición común a los dos posibles modos de funcionamiento consiste en la existencia de una espera del nodo j por el nodo i. El borrado parcial de la espera se corresponde con la precondición que impone

que el estado del nodo i sea activo (figura 3.8.a). Al efecto de eliminar del conjunto $set\_waiters_i$ la información de la espera completa entre el nodo j y el nodo i, (j, t, *received*), le acompaña la inclusión en el mismo conjunto de la terna asociada a la espera en vías de desaparecer, (j, t, *released*) (figura 3.8.b). Si, por el contrario, se analiza la otra precondición para la ejecución de la acción $StartDelArc_i(j,\ t)$, el estado del nodo j será *aborted* (figura 3.8.c). El efecto de la acción en este supuesto se reduce a suprimir la información de la espera de $set\_waiters_i$, (j, t, *received*), que tras la ejecución de $Abort_j$ ya no tiene sentido puesto que el nodo j ha dejado de estar bloqueado por el nodo i (figura 3.8.d). No hay ningún otro efecto más porque en esa situación los efectos de la acción $StartDelArc_i(j,\ t)$ solamente tratan de devolver al sistema un estado consistente tras el aborto de un nodo.

Acción $EndDelArc_i(j)$:

Esta acción se ejecuta en dos situaciones distintas. En un funcionamiento normal, los efectos de esta acción permiten que el nodo i quede finalmente activo, una vez se ha roto la espera que los bloqueaba (figura 3.9.a). Estos cambios vienen acompañados por la actualización del instante de tiempo en el que el nodo ha vuelto a ser activo. En lo que respecta a la variable $set\_waiters_j$, al ejecutarse la acción se eliminará todo rastro de la espera eliminada (figura 3.9.b).

Sin embargo, la acción $EndDelArc_i(j)$ puede ser ejecutada en otra situación que no tiene nada que ver con el funcionamiento normal del sistema ya explicado. Cuando el estado del nodo j es *aborted*, puede ser necesario eliminar esperas contenidas en $set\_waiters_j$ que se iniciaron antes (figura 3.9.c y 3.9.d) o después de la ejecución de la acción $Abort_j$ (figura 3.9.e y 3.9.f), pero que no llegaron a completarse.

En el caso de que el nodo i se bloqueara por el nodo j en estado *aborted*, la espera nunca podría finalizarse porque la acción $EndAddArc_i(j,\ t)$ impide que un nodo en estado *aborted* concluya una espera en formación.

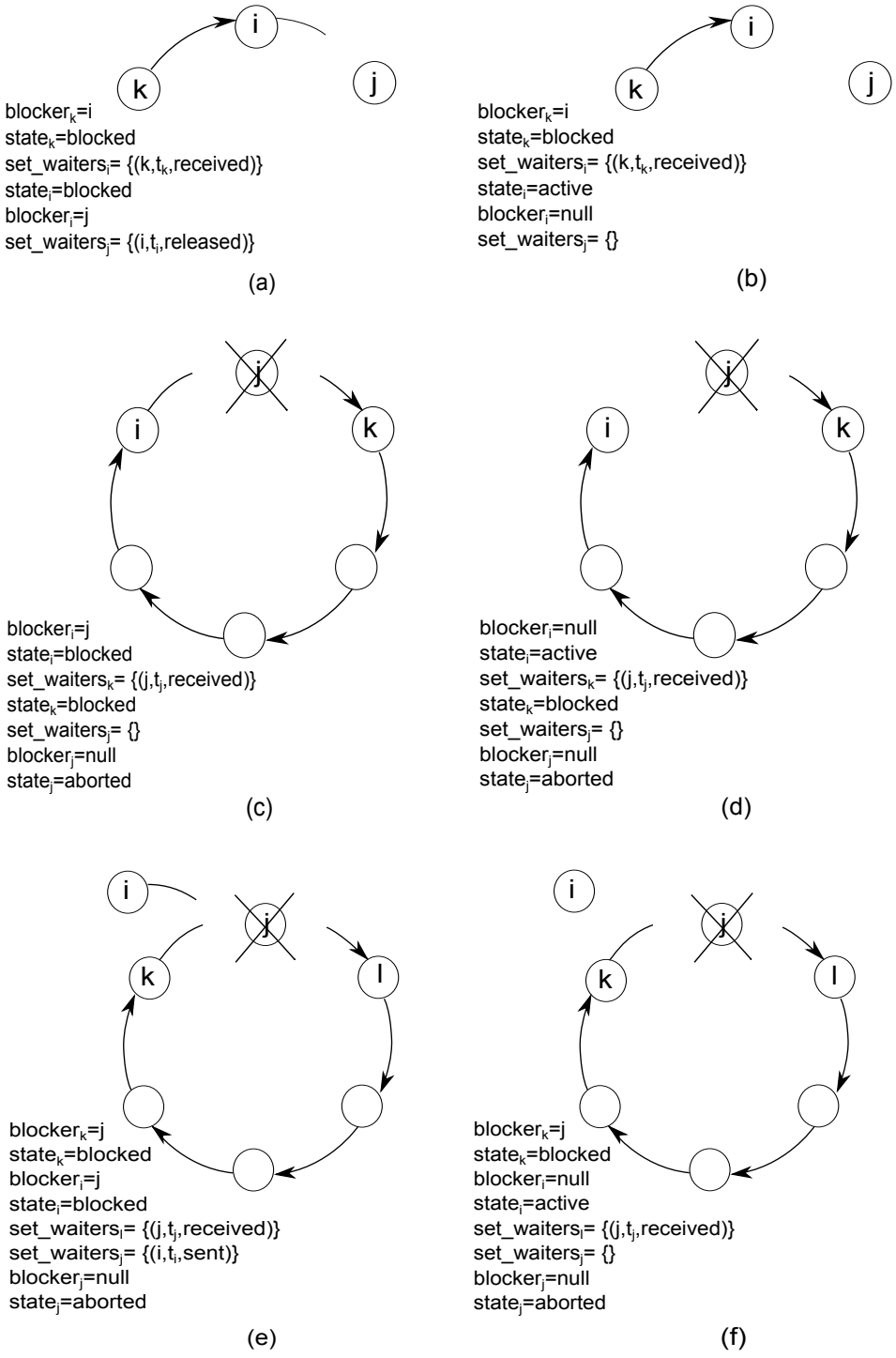

Figura 3.9: Precondiciones (a)-(c)-(e) y efectos (b)-(d)-(f) de la acción $EndDelArc_i(j)$

Las esperas a las que se alude en estos supuestos están marcadas con el estado *sent* (figura 3.9.e). Ese estado determina que el nodo j aún no considera al nodo i como uno de sus predecesores o nodos bloqueados y, al mismo tiempo, el nodo i ignora esa situación. En otras palabras, el nodo i ha iniciado el proceso de creación de una espera quedando bloqueado por el nodo j mientras que el nodo j ni tiene ni puede tener constancia de ese bloqueo.

Los efectos de la acción $EndDelArc_i(j)$, en cualquiera de sus posibles ejecuciones, no impiden que el nodo i pueda volver a bloquearse y a esperar por el nodo j en un instante posterior de la evolución del sistema. Obviamente, esa nueva edición de la espera de i por j tendrá asociada un tiempo posterior al de la espera que se eliminó con esta acción. Esto es cierto incluso cuando $state_j = aborted$. Aunque el nodo j, una vez abortado, no participa en la evolución del sistema, el nodo j no se opone a que se bloqueen por él pero nunca dará por concluida la formación de esa nueva espera.

Acción $Abort_i$:

Esta acción no tiene precondiciones porque se asume que es el efecto de un agente externo al sistema (figura 3.10.a). Los efectos de esta acción consisten en un cambio de estado del nodo i y un incremento de su tiempo de activación para impedir que siendo i un nodo abortado pudiera tener habilitada alguna otra acción del autómata. Por otra parte, se asegura que el nodo abortado ya no espera por ningún nodo del sistema haciendo que $blocker_i$ pase a contener el valor *NULL*. Tras la ejecución de la acción $Abort_i$, también se elimina totalmente la información relativa a las esperas, en cualquiera de sus fases, de todos los nodos que estaban bloqueados por el nodo i. Esa información, almacenada en $set\_waiters_i$, ya no tiene sentido mantenerla tras el aborto del nodo i. Además de los cambios ya mencionados, a la variable $state_i$ se le asigna un nuevo valor, *aborted*, reflejando así el nuevo estado del nodo i (figura 3.10.b).

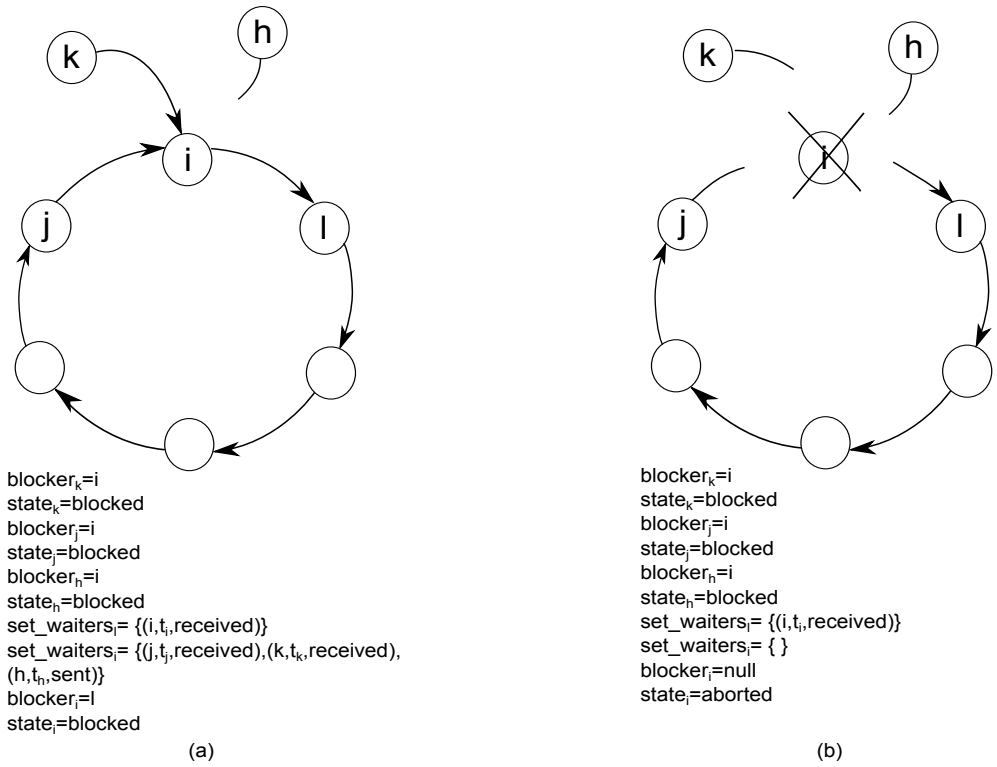

Figura 3.10: Efectos de la acción $Abort_i$.

Partición de G:

La partición de G consta de dos clases de equivalencia tal y como se observa en la figura 3.11. La acción $StartAddArc_i(j)$ constituye una clase por separado para garantizar el bloqueo de algún nodo del sistema. El resto de acciones del autómata forman parte de otra clase permitiendo así que una relación de espera se complete y que las esperas desaparezcan de sistema.

$$\texttt{Part}(G) = \{\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j}\texttt{:}\ \{\texttt{StartAddArc}_i\texttt{(j)}\},\ \forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{t} \in \mathbb{N}\texttt{:}\ \{\texttt{EndAddArc}_i\texttt{(j,t)},\ \texttt{StartDelArc}_i\texttt{(j,t)},\ \texttt{EndDelArc}_i\texttt{(j)}\}\}$$

Figura 3.11: Partición del autómata del medio, G.

3.1.2.3. Modelo *Request-Grant-Release-Withdraw*

Como el autómata que modela el sistema gestor puede parecer demasiado abstracto, es conveniente mostrar que es capaz de imitar el funcionamiento de otros modelos que aparecen en la literatura. Se va a considerar el modelo *Request-Grant-Release-Withdraw* de los sistemas gestores de [56] y [55] que emplean gestores de procesos y recursos distribuidos con el modelo de petición de único recurso.

Este modelo diferencia entre procesos y recursos. Cuando el proceso p necesita el recurso r, lo solicita mediante la acción $request_p(r)$. Esta acción equivale al envío de un mensaje de petición de p a r. Cuando el recurso recibe la petición o bien el gestor del recurso le asigna el recurso r a p o bien, en caso de que el recurso esté retenido por otro proceso, el proceso p queda a la espera hasta que esté libre. La acción de asignación, $grant_r(p)$, consiste en la entrega de un mensaje que garantiza que el recurso r está asignado al proceso p. Posteriormente, el proceso p puede liberar el recurso r que tiene retenido. Para ello, se ejecuta la acción $release_p(r)$ que conlleva la transmisión de un mensaje de p al recurso r. La última acción que falta por describir es la que el gestor debe realizar cuando el proceso p ha abortado. Esta acción $withdraw_p(r)$ se encarga de cancelar cualquier solicitud pendiente a un recurso r que no hubiera sido atendida en el momento de producirse el aborto del proceso p.

El modelo del sistema empleado permite simular el funcionamiento de las acciones de un sistema gestor con el principio de operación *Request-Grant-Release-Withdraw*. Las variables del autómata permiten que no sea necesaria la aparición explícita de los mensajes enviados en las instancias del gestor.

La solicitud de un recurso r por parte de un proceso p requiere que el proceso p esté activo inicialmente. En el modelo de sistema que está siendo usado, esto equivale a que $state_p$ sea *active* y que no tenga otras solicitudes pendientes, $blocker_p$ no tenga ningún nombre de recurso anotado ($NULL$). La solicitud de p por el recurso r se modela mediante la ejecución, en primer lugar, de la acción $StartAddArc_p(r)$, cambiando

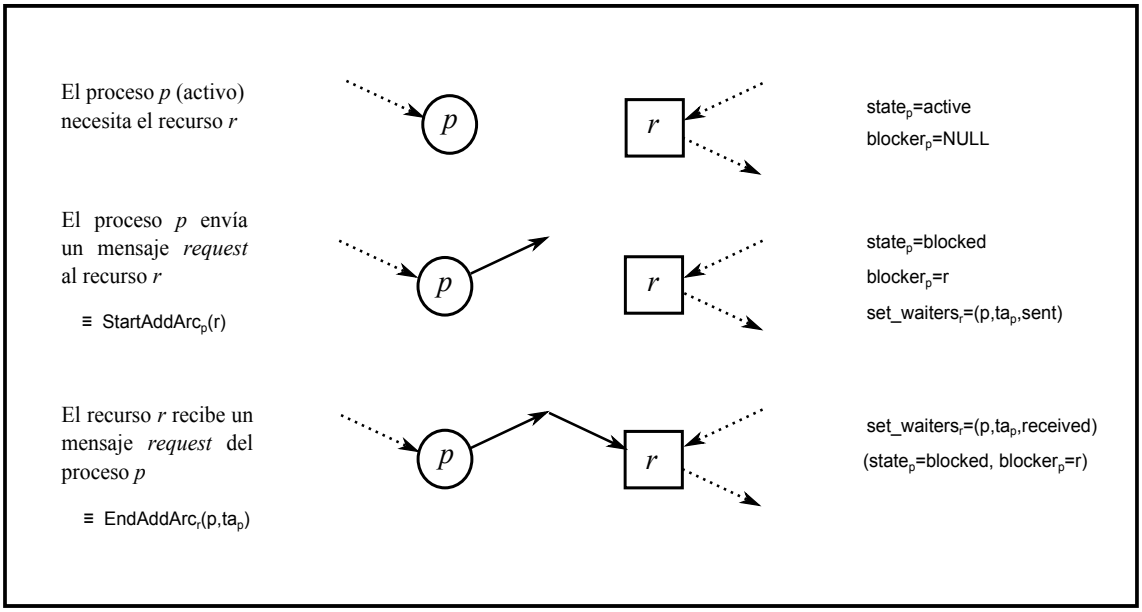

Figura 3.12: Secuencia de solicitud de un recurso: $request_p(r)$.

el estado del proceso p a *blocked* e incluyendo el nombre del recurso como $blocker_p = r$. Los efectos de esta acción incorporan al conjunto $set\_waiters_r$ la tupla (p, t_activ_p, *sent*), lo cual implica que el proceso p en el instante t_activ_p ha enviado, *sent*, una solicitud por el recurso r. Tras la ejecución completa de esta acción, el proceso p queda a la espera de r, pero el recurso r todavía desconoce la solicitud del proceso p. Para concluir el proceso de solicitud, el recurso dará cuenta de la recepción de la petición del proceso p ejecutando la acción $EndAddArc_r(p, t)$, que simula la recepción del mensaje *request*. El único efecto que producirá esta acción será el de sustituir la tupla (p, t, *sent*) del conjunto $set\_waiters_r$ por una nueva tupla (p, t, *received*) que señala que el recurso r es consciente de la solicitud del proceso p. En la figura 3.12 se puede observar esta operación completa.

Del mismo modo, la fase de asignación de un recurso puede simularse en el modelo de sistema que se ha definido. Dando por hecho que el proceso p está bloqueado esperando por r, que el recurso r es conocedor de esta espera y además el recurso r

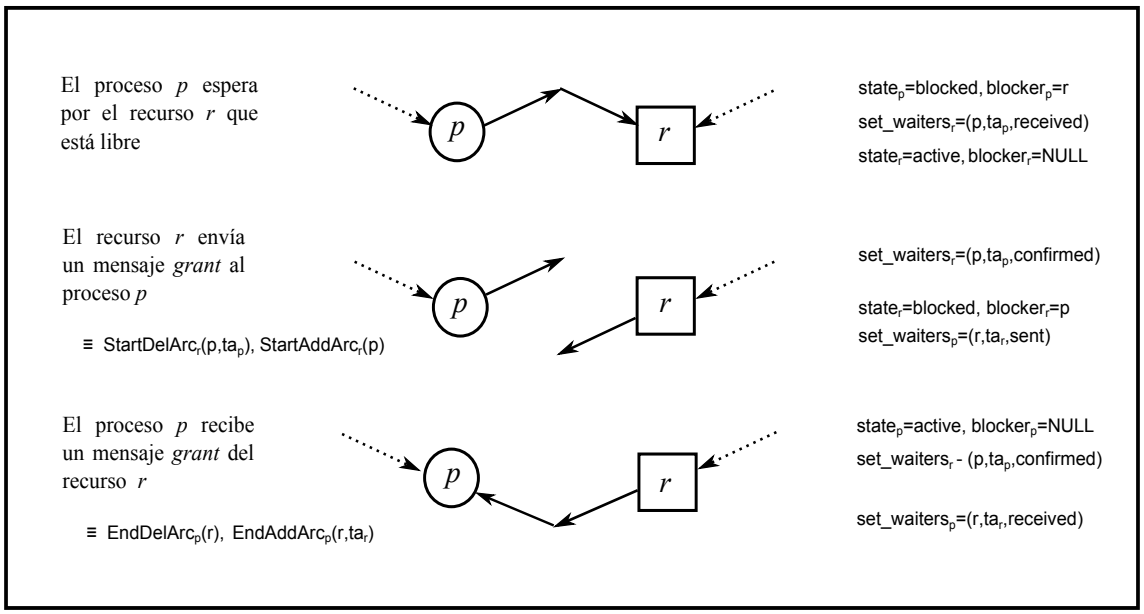

Figura 3.13: Secuencia de asignación de un recurso: $grant_r(p)$.

está libre, se puede proceder a su asignación al proceso p. Todas esas condiciones se corresponden en el modelo utilizado a: $state_p = blocked$, $blocker_p = r$, $(p, t, received) \in set\_waiters_r$ y $state_r = active$.

El envío del mensaje *grant* se modela mediante la acción $StartDelArc_r(p, t)$ seguida de la acción $StartAddArc_r(p)$. Los efectos de la primera acción modifican el conjunto $set\_waiters_r$ eliminando la tupla $(p, t, received)$ añadiendo en su lugar la tupla $(p, t, released)$, que significa que la solicitud se va a atender. El recurso ejecuta la acción $StartAddArc_r(p)$ que hace que $state_r$ sea *blocked* y ningún otro proceso pueda acceder a él. Además, $blocker_r = p$ y en $set\_waiters_p$ se incluye la tupla $(r, t\_activ_r, sent)$ con objeto de dar a conocer al proceso p que el recurso r está asignado a él desde el instante t_activ_r.

A continuación, se deben ejecutar las acciones $EndDelArc_p(r)$ y $EndAddArc_p(r, t)$ que simulan la recepción del mensaje *grant* (ver figura 3.13). La primera acción suprime la información que resta de la solicitud del recurso r que lanzó el proceso

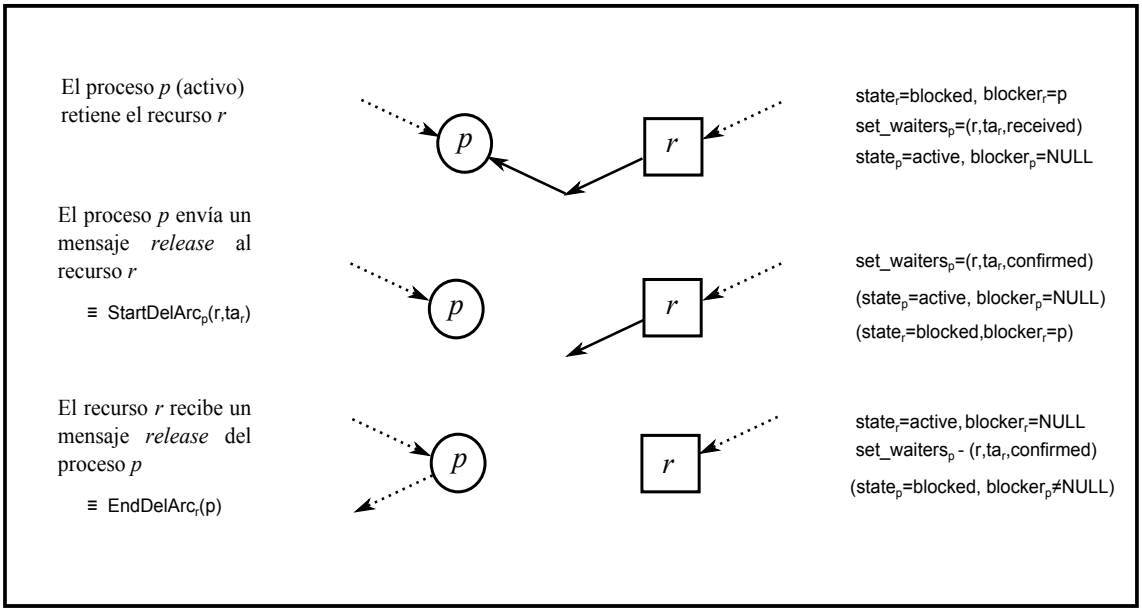

Figura 3.14: Secuencia de liberación de un recurso: $release_p(r)$.

p y la segunda permite que p conozca que r espera su liberación. Los efectos de $EndDelArc_p(r)$ dejan al proceso en condiciones de hacer una nueva solicitud de recurso, es decir, $state_p$ vuelve a ser *active* en un nuevo tiempo de activación, t_activ_p+1, y $blocker_p$ ya no guarda ningún nombre de recurso ($NULL$). Al mismo tiempo, se suprime la tupla (p, t, *released*) de $set\_waiters_r$ porque la confirmación de la asignación ya está asumida. Con la acción $EndAddArc_p(r,\ t)$ se da por terminado el proceso de asignación ya que $set\_waiters_p$ cambia la tupla (r, t, *sent*) por (r, t, *received*).

Así mismo, la realización de la fase de liberación de un recurso está asegurada con la ejecución de las acciones $StartDelArc_p(r,\ t)$ y $EndDelArc_r(p)$ en ese orden (véase la figura 3.14). Partiendo del hecho de que un proceso activo p tiene asignado un recurso r, esto es, $state_p = active$, $blocker_r = p$, $(r,\ t,\ received) \in set\_waiters_p$, y $state_r = blocked$, se ejecuta en primer lugar la acción $StartDelArc_p(r,\ t)$. El efecto de esta acción es fundamentalmente que el proceso p deje de retener el recurso r. Para eso, se reemplaza la tupla (r, t, *received*) de $set\_waiters_p$ por (r, t, *released*). Una vez liberado

el recurso por parte del proceso p, la acción $EndDelArc_r(p)$ hace que el recurso sea conocedor de que es libre de nuevo. Los cambios producidos por los efectos de esta acción son: $state_r = active$, $blocker_r = NULL$, t_activ_r+1 y se elimina $(r, t, released)$ de $set\_waiters_p$.

Cuando aborta un proceso p, $kill(p)$, se sucede una secuencia de acciones para liberar los recursos que tenía asignados y, al mismo tiempo, se produce otra secuencia de acciones que anula la petición del recurso por el que estaba esperando, $withdraw_p(r)$. La liberación de un recurso asignado a p tras su aborto es diferente a la liberación de un recurso asignado que se ha descrito anteriormente. En el caso de la liberación de un recurso tras el aborto del proceso al que estaba asignado, sólo se precisa que se realice la última fase en la que el recurso quedaba libre de nuevo (ver figura 3.15).

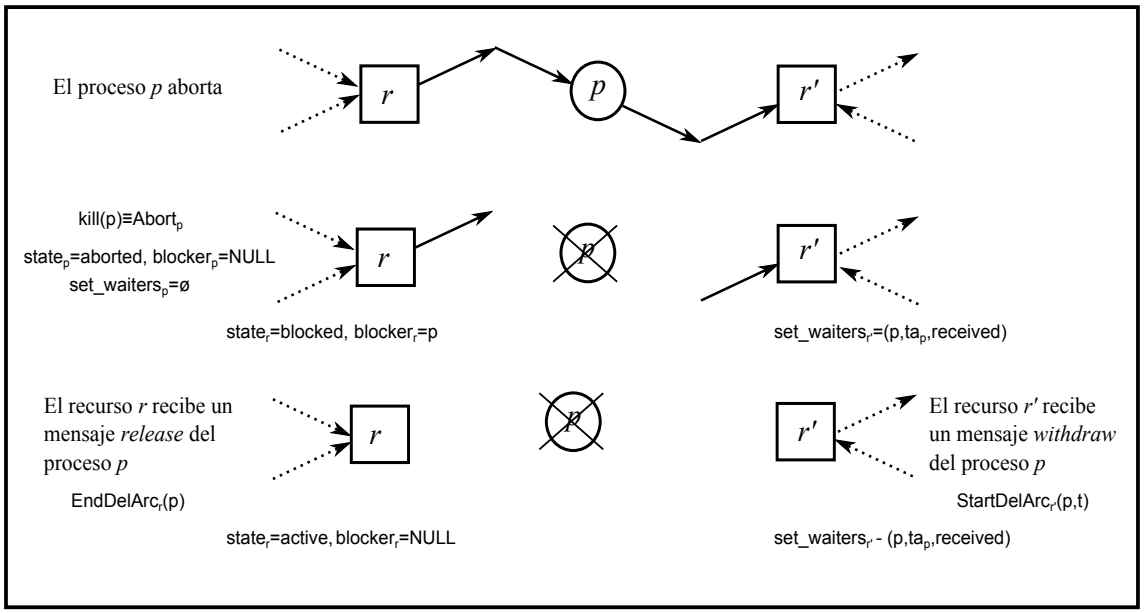

Figura 3.15: Secuencia del aborto de un proceso: $kill(p)$.

En el autómata que se ha construido para modelar el sistema, este efecto se consigue con la ejecución de la acción $EndDelArc_r(p)$. La fase inicial de la liberación que se realizaba mediante la acción $StartDelArc_p(r, t)$ no debe ejecutarse porque la acción

$Abort_p$, equivalente a $kill(p)$, adelanta este proceso. Para anular la petición de un recurso que lanzó un proceso que ya ha abortado se dispone de la acción $withdraw_p(r)$. Esta acción es asimilable a la acción del autómata $StartDelArc_r(p,\ t)$. Tanto la acción $EndDelArc_r(p)$ como la acción $StartDelArc_r(p,\ t)$ incorporan en la condición de habilitación la situación de que el proceso p esté abortado e identifican los efectos que corresponden a esa precondición.

Tras el aborto de un proceso pueden surgir comportamientos anómalos. Es posible, por ejemplo, que se ejecuten concurrentemente la acción que hace que el proceso p aborte y la asignación de un recurso al proceso p. Si el proceso p ha recibido el mensaje de asignación de un recurso después de que el proceso p aborte, se tendrá que realizar la liberación del recurso de igual forma que para el resto de recursos asignados. Si el proceso de la asignación del recurso se ha iniciado a la par que se ejecuta la acción $Abort_p$, en este caso la anulación se realizará mediante la acción $withdraw_p(r)$ y se desarrollará por la acción del autómata $EndDelArc_r(p)$.

Aunque hay fases de las distintas acciones del modelo que se representan gráficamente del mismo modo, el estado del sistema es diferente. Si se considera el estado del autómata de cada una de esas fases en las que el grafo del sistema es idéntico, se aprecia fácilmente que los valores de las variables permiten distinguirlas. Las variables del autómata son las que reflejan el envío y la recepción de los mensajes entre nodos del sistema que el gestor organiza y supervisa.

El autómata del sistema que se ha analizado recoge tanto el comportamiento de procesos como de recursos. Este es el principal motivo por el que será habitual denominar nodos a todos los elementos del sistema y no distinguir entre sus funcionalidades.

3.2. Caracterización formal del interbloqueo

Los valores de las variables que definen el estado del autómata S_0 permiten conocer con exactitud la situación en la que se halla el sistema y, por tanto, si en un estado dado hay un interbloqueo o no. Para concluir que en el sistema existe un interbloqueo será necesario analizar las relaciones de espera existentes entre los nodos del sistema, pero ésa no es una tarea inmediata.

En este apartado se pretende caracterizar un interbloqueo empleando las variables que definen el estado del sistema. Con el fin de establecer de una forma rigurosa cómo se relacionan las esperas entre los nodos y un interbloqueo de los mismos, también se definen una serie de conceptos que facilitan la comprensión y la escritura del formalismo utilizado según la información del medio.

Definición 3.2.1. *Espera directa entre dos nodos adyacentes en el grafo*, $dwait(i, j, s)$. Se dice que, en un estado $s \in states(S_0)$, hay una espera directa entre el nodo i y el nodo j, denotado como $dwait(i, j, s)$, si y sólo si:

- $s.state_i = blocked$
- $s.blocker_i = j$
- $(i, s.t\_activ_i, received) \in s.set\_waiters_j$

Dada la espera directa $dwait(i, j, s)$, se puede establecer que, en el estado s, el nodo i es un predecesor inmediato o directo del nodo j y el nodo j es un sucesor inmediato o directo del nodo i.

Las demostraciones de diversas propiedades del algoritmo, que aparecerán en capítulos posteriores, emplean mayoritariamente el concepto de ruta. A diferencia de un camino, en una ruta cada elemento consta de dos componentes: el identificador del nodo que forma parte del camino que describe la ruta, y el tiempo de activación

de ese nodo en el instante que se registró en la ruta. Antes de comentar cómo se opera con las rutas, se proporcionan las definiciones necesarias que formalizan el concepto.

Definición 3.2.2. Se denomina P al conjunto de posibles pares (*identificador nodo*, *tiempo de activación nodo*) formado a partir de los nodos del sistema, esto es, $P = \{(n,\, t)\colon n \in \mathcal{N} \wedge t \in \mathbb{N}\}$.

Definición 3.2.3. Se llama P^+ al conjunto de posibles rutas que puede aparecer en una ejecución del autómata del sistema, o lo que es lo mismo, cualquier concatenación de elementos del conjunto P. Formalmente se expresa como: $P^+ = \{(n_1,\, t_1)\ (n_2,\, t_2)\ \ldots (n_i,\, t_i)\colon \forall\ j,\, 1 \leq j \leq i,\, n_j \in \mathcal{N} \wedge t_j \in \mathbb{N}\}$.

El concepto de espera directa se puede ampliar para crear el concepto de espera transitiva o espera indirecta entre dos nodos no adyacentes en el grafo.

Definición 3.2.4. *Espera transitiva o camino entre dos nodos del grafo*, $wait(i,\, j,\, p,\, s)$. Se dice que, en un estado $s \in states(S_0)$, hay una espera indirecta entre un nodo $i \in \mathcal{N}$ y un nodo $j \in \mathcal{N}$ a través de una ruta $p = (n_1,\, t_1)(n_2,\, t_2)\ \ldots\ (n_m,\, t_m)$, denotado como $wait(i,\, j,\, p,\, s)$, si y sólo si:

- $n_1 = i \wedge n_m = j$
- $\forall\ k\colon 1\ \ldots\ m{-}1,\ dwait(n_k,\, n_{k+1},\, s)$
- $\forall\ k\colon 1\ \ldots\ m{-}1,\ t_k = s.t\_activ_{n_k}$

En la espera transitiva $wait(i,\, j,\, p,\, s)$, se considera que el nodo i es un predecesor (no inmediato) del nodo j y el nodo j es un sucesor (no inmediato) del nodo i. Seguidamente, se define una función que permite hacer referencia a los nodos que componen un camino.

Definición 3.2.5. *Nodos de una ruta,* $nodes(p)$. Dada una ruta p que verifica $wait(1,\, l,\, p,\, s)$, al aplicar la función $nodes(p)$ se obtiene la secuencia de nodos $(n_1,\, n_2,\, \ldots,$

n_l). El resultado de esta función es evidente si p se representa como una secuencia de esperas directas entrelazadas (espera transitiva).

Para extraer información de las rutas existentes en las ejecuciones del autómata, se emplean una serie de funciones definidas sobre el conjunto P^+ de la siguiente manera:

- *first*: $P^+ \rightarrow \{(n_i,\ t_i)\colon n_i \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Dada la ruta $p = (n_1,\ t_1)\ (n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$, la función *first*$(p)$ obtiene el primer elemento de p, esto es, $(n_1,\ t_1)$.

- *penult*: $P^+ \rightarrow \{(n_i,\ t_i)\colon n_i \in \mathcal{N} \wedge t \in \mathbb{N}\} \cup NULL$. Sea p, la ruta $(n_1,\ t_1)\ (n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$, la función *penult*$(p)$ permite extraer el penúltimo elemento de p. Si la ruta cuenta con más de un elemento, $k > 1$, el resultado de la función es $(n_{k-1},\ t_{k-1})$. En caso contrario, $k \leq 1$, la función devuelve el valor $NULL$, con el que se indica que la ruta no dispone del mínimo número de elementos para aplicar la función.

- *last*: $P^+ \rightarrow \{(n_i,\ t_i)\colon n_i \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Dada la ruta $p = (n_1,\ t_1)\ (n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$, la función *last*$(p)$ da como resultado el último elemento de p, esto es, $(n_k,\ t_k)$.

- *visited_nodes*: $P^+ \rightarrow 2^P$. Sea la ruta $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$, al aplicar esta función sobre p se obtienen todos los pares (nodo, tiempo) que componen p, *visited_nodes*$(p,\ s) = \{(n_i,\ t_i)\colon 1 \leq i \leq k\}$.

- operador resta $-$: $P^+ \rightarrow P^+$. Sea la ruta $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$, al realizar esta operación se eliminan el par o pares (nodo, tiempo) que se indican tras el signo "$-$". Por ejemplo, $p -$ *first*(p) da como resultado la ruta $(n_2,\ t_2)\ \ldots\ (n_k,\ t_k)$.

Después de haber introducido el concepto de ruta, se puede presentar el concepto de ciclo, básico en teoría de grafo, y que se asimila al de interbloqueo. Un ciclo es un

camino, que parte de un nodo y finaliza en él mismo, es decir, se trata de una espera en la que un nodo espera en forma transitiva por si mismo.

Definición 3.2.6. *Ciclo.* Se dice que la ruta $p \in P^+$ es un ciclo si y sólo si $wait(i, j, p, s) \wedge i = j$.

Resulta evidente que en un ciclo todos los nodos que lo componen esperan transitivamente por sí mismos debido a la simetría propia de un ciclo. Así que, para referirse a la existencia de un ciclo se usará frecuentemente la notación $\mathcal{C}(p, s)$, en la que no se señala ningún nodo como inicio (y final) del camino considerado y en la que se asume que la ruta p es el ciclo propiamente dicho.

Definición 3.2.7. *Ciclos en un sistema de $\mathcal{N}$ nodos.* La función $cycles(\mathcal{N})$ obtiene los ciclos que pueden formarse en un sistema constituido por $\mathcal{N}$ nodos. Formalmente se puede escribir como $cycles(\mathcal{N})$ que verifica $\{wait(i, i, c, s)$, donde $\exists\ i \in \mathcal{N} \wedge c \in P^+ \wedge \exists\ s \in states(G)\}$.

Definición 3.2.8. *Ciclos en un estado.* Dado un estado s, se define la función $cycles(s)$ como la función que devuelve todos los ciclos existentes en el sistema en el estado s.

En el modelo de único recurso, un nodo espera como máximo por otro nodo y, en consecuencia, todos los ciclos presentes en un sistema en un estado son disjuntos. Un interbloqueo para este tipo de modelo quedará caracterizado por la existencia de un ciclo en el grafo con el que se representan las relaciones de espera entre nodos (recursos y procesos según [14]).

Definición 3.2.9. *Interbloqueo.* En el estado s existe un interbloqueo si y sólo si $\exists$ $\mathcal{C}$ tal que $\mathcal{C} \in cycles(s)$.

3.3. Especificación del problema

3.3.1. Criterios de corrección

Los criterios de corrección que debe cumplir todo algoritmo de detección y resolución de interbloqueos son:

- *Propiedad de seguridad:* Si el algoritmo aborta un proceso es porque éste forma parte de un interbloqueo.

- *Propiedad de viveza:* El algoritmo resuelve todos los interbloqueos en un tiempo finito.

En la primera sección de este capítulo se ha descrito formalmente el funcionamiento del sistema considerando exclusivamente el comportamiento del entorno del algoritmo que es dirigido por el gestor del sistema. La formalización proporcionada gracias al autómata G deja al descubierto que las acciones que controla el sistema son las que producen la aparición y desaparición de arcos en el grafo del sistema. Sin embargo, la acción que provoca el aborto de un nodo del sistema no está bajo el control del sistema. En la definición de la acción $Abort_i$ en G esto queda patente al no haber ninguna precondición para la habilitación de la acción. Se dice en esta situación que el modelo del sistema es libre de abortos espontáneos, lo que lo aleja del comportamiento de un sistema real.

Es muy frecuente que en los trabajos de interbloqueo distribuido se asuma que el sistema está libre de abortos espontáneos, dado que permitiendo su existencia, se violaría la propia condición de seguridad, que se impone para el diseño de un algoritmo de resolución de interbloqueos. La demostración de este resultado de imposibilidad, reformulada con autómatas de Entrada/Salida, se puede encontrar en [15]. Una vez definido por completo el autómata S_0, es posible formalizar los criterios de corrección

enunciados usando el modelo de autómatas de Entrada/Salida y a partir del propio S_0.

3.3.2. Formalización de la condición de seguridad

La condición de seguridad que debe satisfacer el algoritmo requiere de la definición de un nuevo autómata, A_0. Este autómata proporciona la especificación abstracta del algoritmo y sus ejecuciones aseguran que sólo aborten nodos interboqueados. El autómata A_0 debe ser compatible con el autómatata del gestor, G, por lo que las acciones de ambos están íntimamente relacionadas. Así que, conforme se describa el autómata A_0 en esta sección, se remarcarán las semejanzas y diferencias con el autómata del gestor, G.

En primer lugar, hay que mencionar que para el autómata A_0 el estado de las esperas del sistema también puede quedar representado en un grafo. El conjunto de arcos del grafo se podrá definir de igual manera que en el grafo empleado para el autómata del gestor, G.

Estado de A_0:

Las variables que permiten definir el estado de A_0 son similares a las variables que definen el estado de G en la figura 3.1. Un estado $s \in states(A_0)$ viene determinado por los valores que tomen esas variables. En la definición de G se empleó otra notación, pero hay una equivalencia entre variables que hacen referencia al mismo objeto y los valores que se adoptan reflejan un mismo estado.

El estado inicial de A_0 se corresponde con el instante en el que tiene lugar la puesta en marcha del sistema $s_0 \in start(A_0)$ y también es similar al del autómata G (ver figura 3.3).

Signatura de A_0:

La signatura de acciones de A_0 es una signatura externa, es decir, el autómata no cuenta con acciones internas (ver figura 3.16).

$$\begin{aligned}
&\texttt{Input}(A_0) = \{\texttt{StartAddArc}_i\texttt{(j)}, \texttt{EndAddArc}_i\texttt{(j,t)}, \texttt{StartDelArc}_i\texttt{(j,t)}, \texttt{EndDelArc}_i\texttt{(j)} \\
&\qquad\qquad \texttt{tal que } \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \texttt{t} \in \mathbb{N}\} \\
&\texttt{Internal}(A_0) = \emptyset \\
&\texttt{Output}(A_0) = \{\texttt{Abort}_i \texttt{ tal que i} \subseteq \mathcal{N}\}
\end{aligned}$$

Figura 3.16: Signatura de acciones del autómata del algoritmo, A_0.

Acciones de A_0:

Las acciones de A_0, del mismo modo que las acciones de G, representan las modificaciones del grafo. Su significado es el mismo que el de las acciones de G, es decir, $StartAddArc_i(j)$ y $EndAddArc_i(j, t)$ reflejan la aparición de una espera, $StartDelArc_i(j, t)$ y $EndDelArc_i(j)$ describen la desaparción de una espera y $Abort_i$ informa del aborto de nodos.

En la figura 3.17 se muestran los pasos del autómata A_0 mediante el esquema de precondición-efecto. Puede observarse que las acciones internas carecen de precondición, pero la acción de salida $Abort_i$ incluye como condición la existencia de un ciclo en el que el nodo i será el que resuelva el interbloqueo.

```
StartAddArc_i(j)
precondiciones:
efectos:
 state_i := blocked;
 blocker_i := j;
 set_waiters_j := set_waiters_j ∪ {(i, t_activ_i, sent)}.

EndAddArc_i(j,t)
precondiciones:
efectos:
 set_waiters_i := set_waiters_i ∪ {(j, t, received)} \ {(j, t, sent)}.

StartDelArc_i(j,t)
precondiciones:
efectos:
 set_waiters_i := set_waiters_i \ {(j, t, received)};
 IF (state_i = active) THEN
  set_waiters_i := set_waiters_i ∪ {(j, t, released)}
 ENDIF.

EndDelArc_i(j)
precondiciones:
efectos:
 IF (state_j ≠ aborted) THEN
  set_waiters_j := set_waiters_j \ {(i, t_activ_i, released)}
 ELSEIF ((i, t_activ_i, sent) ∈ set_waiters_j) THEN
  set_waiters_j := set_waiters_j \ {(i, t_activ_i, sent)}
 ENDIF;
 state_i := active;
 blocker_i := NULL;
 t_activ_i := t_activ_i + 1;

Abort_i
precondiciones: ∃ p ∈ P^+: C(p,s) o wait(i,i,p,s).
efectos:
 state_i := aborted;
 blocker_i := NULL;
 t_activ_i := t_activ_i + 1;
 set_waiters_i := ∅.
```

Figura 3.17: Acciones del autómata del algoritmo, A_0.

Partición de A_0:

Conocida la signatura y las acciones del autómata A_0, se muestra la partición del mismo en la figura 3.18.

$$\texttt{Part}(A_0) = \{\forall\ \texttt{i} \in \mathcal{N}\texttt{: } \{\texttt{Abort}_i\}\}$$

Figura 3.18: Partición del autómata del algoritmo, A_0.

Observando las definiciones de los autómatas A_0 y G se aprecia que ambos autómatas son fuertemente compatibles. Esta relación de compatibilidad se basa en las siguientes propiedades:

- $Internal(G) = Internal(A_0) = \emptyset \Rightarrow$

  $Internal(G) \cap acts(A_0) = Internal(A_0) \cap acts(G) = \emptyset$

- $Output(G) \cap Output(A_0) = Output(G) \cap Input(G) = \emptyset$

Es bien sabido que el gestor del sistema se encarga de controlar la petición de recursos y su asignación y, por consiguiente, se puede decir que el gestor controla directamente la creación y desaparición de esperas. Como es el gestor el que proporciona la información del estado de las esperas al algoritmo, queda justificado que las acciones de entrada de A_0 sean las acciones de salida de G. Por el contrario, las acciones de salida de A_0 se corresponden con las acciones de entrada de G. Esto se debe a que, una vez que el algoritmo lleva a cabo sus funciones, es el propio algoritmo el que informa al gestor, mediante la acción $Abort_i$, de la identificación de la víctima que resuelve el interbloqueo detectado. Finalmente, es el gestor el que fuerza el aborto de esa víctima y refleja en el sistema los efectos de ese aborto. La composición de los autómatas A_0 y G da lugar a un nuevo autómata que se llamará S_0 (ver figura 3.19) y que realmente describe el funcionamiento de un sistema libre de interbloqueos.

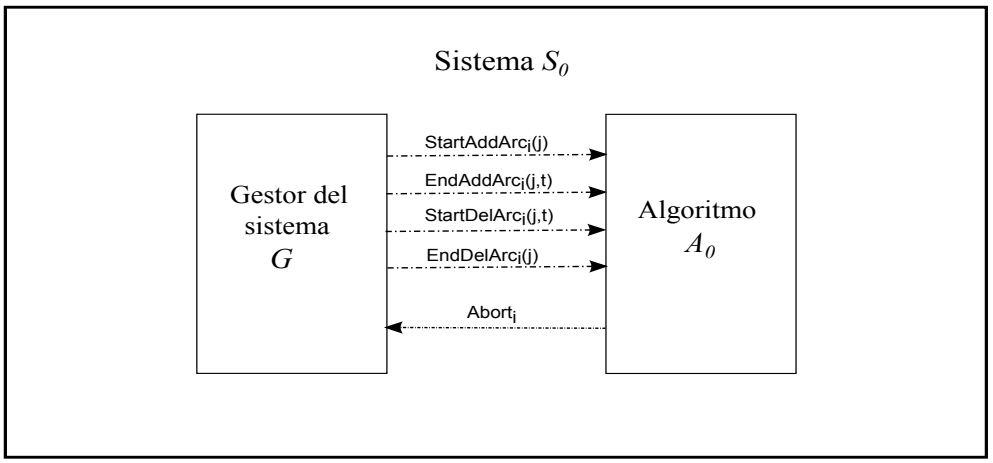

Figura 3.19: Composición del autómata G y del autómata A_0.

Estado de S_0:

Aplicando propiedades de la composición de autómatas, se concluye que el estado del autómata S_0, resultado de la composición del autómata del gestor, G, y del autómata del algoritmo, A_0, está formado por la unión de de los conjuntos de variables de los autómatas componentes. Por otra parte, la semejanza de las acciones y sus efectos en ambos autómatas va a permitir simplificar la notación referida al autómata S_0. Como las variables que determinan un estado alcanzable por S_0 se denotan de igual manera en los autómatas G y A_0 se evitará la duplicidad de variables en la definición de ese estado. El estado inicial de S_0 se corresponde con la puesta en marcha del sistema, $s_0 \in start(S_0)$. Las variables y los valores que les corresponden aparecen en la figura 3.20.

Estado inicial (s_0) del autómata del sistema S_0:
$\forall$ i $\in \mathcal{N}$: s_0.state$_i$ = active
$\forall$ i $\in \mathcal{N}$: s_0.t_activ$_i$ = 1
$\forall$ i $\in \mathcal{N}$: s_0.blocker$_i$ = NULL
$\forall$ i $\in \mathcal{N}$: s_0.set_waiters$_i$ = $\emptyset$

Figura 3.20: Estado inicial del autómata del sistema, S_0.

```
StartAddArc_i(j)
precondiciones:
efectos:
 state_i := blocked;
 blocker_i := j;
 set_waiters_j := set_waiters_j ∪ {(i, t_activ_i, sent)}.

EndAddArc_i(j,t)
precondiciones:
efectos:
 set_waiters_i := set_waiters_i ∪ {(j, t, received)} \ {(j, t, sent)}.

StartDelArc_i(j,t)
precondiciones:
efectos:
 set_waiters_i := set_waiters_i \ {(j, t, received)};
 IF (state_i = active) THEN
  set_waiters_i := set_waiters_i ∪ {(j, t, released)}
 ENDIF.

EndDelArc_i(j)
precondiciones:
efectos:
 IF (state_j ≠ aborted) THEN
  set_waiters_j := set_waiters_j \ {(i, t_activ_i, released)}
 ELSEIF ((i, t_activ_i, sent) ∈ set_waiters_j) THEN
  set_waiters_j := set_waiters_j \ {(i, t_activ_i, sent)}
 ENDIF;
 state_i := active;
 blocker_i := NULL;
 t_activ_i := t_activ_i + 1;

Abort_i
precondiciones: ∃ p ∈ P^+: C(p,s) o wait(i,i,p,s).
efectos:
 state_i := aborted;
 blocker_i := NULL;
 t_activ_i := t_activ_i + 1;
 set_waiters_i := ∅.
```

Figura 3.21: Acciones del autómata del sistema, S_0.

Acciones de S_0:

El conjunto $steps(S_0)$ se construye a partir de los correspondientes conjuntos de G y A_0. Asumiendo que en un estado alcanzable de S_0 las variables de los autómatas son equivalentes, las acciones del autómata S_0 son las mismas que se definieron para el autómata G a excepción de la acción $Abort_i$ (figura 3.21). Como consecuencia de la composición de G y A_0, la acción $Abort_i$ de S_0 adopta como precondición la que

aparece en la misma acción del autómata A_0.

Signatura de S_0:

Dado que el autómata del gestor, G, y el autómata del algoritmo, A_0, presentan signaturas externas de acciones, la composición de ambos autómatas determina que la signatura de acciones del autómata resultante, S_0, sólo contenga acciones de salida (ver figura 3.22).

$$\begin{aligned}
&\texttt{Input}(S_0) = \texttt{Internal}(S_0) = \emptyset \\
&\texttt{Output}(S_0) = \{\forall\ \{\texttt{i},\texttt{j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{t} \in \mathbb{N}\texttt{: StartAddArc}_i\texttt{(j), EndAddArc}_i\texttt{(j,t),} \\
&\qquad\qquad \texttt{StartDelArc}_i\texttt{(j,t), EndDelArc}_i\texttt{(j);} \\
&\qquad\qquad \forall\ \texttt{i} \in \mathcal{N}\texttt{: Abort}_i\}
\end{aligned}$$

Figura 3.22: Signatura de las acciones del autómata S_0.

Del autómata definido en la figura 3.21 se puede asegurar que sus ejecuciones verifican la condición de seguridad. Las propiedades de autómatas de Entrada/Salida establecen que el autómata resultante de la composición de dos autómatas preserva las propiedades de los autómatas involucrados en la operación. Por consiguiente, el autómata S_0 cumple todas las propiedades del autómata G. Esto implica que si aparece un ciclo en el grafo asociado a un estado de S_0, ese ciclo representa la existencia de un interbloqueo en el sistema.

Por otro lado, la acción $Abort_i$ de S_0 queda habilitada, al igual que en la acción homónima de A_0, si hay un ciclo en el sistema. Considerando ambas situaciones se puede concluir que en las ejecuciones de S_0 sólo se admite que aborten nodos que formen parte del interbloqueo.

Partición de S_0:

La partición de un autómata permite establecer conjuntos de acciones cuya ejecución verifica propiedades de equidad. Una ejecución de un autómata es equitativa

cuando, dada una clase que está continuamente habilitada, finalmente se ejecuta alguna de sus acciones. Asumiendo que las ejecuciones del autómata son equitativas, se podrán verificar ciertas propiedades de viveza.

La figura 3.23 muestra la partición del autómata del sistema, S_0. La definición de la partición asegura el progreso del algoritmo de detección y resolución de interbloqueo propuesto como solución. No es objetivo de este trabajo garantizar que los procesos ejecutados progresen adecuadamente.

$$\begin{aligned}\texttt{Part}(S_0) = \{\ & \forall\ \{\texttt{i},\texttt{j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j}\texttt{:}\ \{\texttt{StartAddArc}_i\texttt{(j)}\},\\ & \forall\ \{\texttt{i},\texttt{j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \texttt{t} \in \mathbb{N}\texttt{:}\ \{\texttt{EndAddArc}_i\texttt{(j,t)},\ \texttt{StartDelArc}_i\texttt{(j,t)},\ \texttt{EndDelArc}_i\texttt{(j)}\},\\ & \forall\ \texttt{i} \in \mathcal{N}\texttt{:}\ \{\texttt{Abort}_i\}\ \}\end{aligned}$$

Figura 3.23: Partición del autómata S_0.

3.3.3. Formalización de la condición de progreso

Las ejecuciones de S_0 verifican la condición de seguridad, pero es necesario asegurar que el algoritmo progresa de forma correcta. Para poder asegurar que ningún interbloqueo permanece indefinidamente, el sistema tiene que cumplir ciertas propiedades de equidad. En el modelo de Autómatas de Entrada/Salida, la propiedad de equidad que se asume indica que no puede haber una clase de equivalencia de la partición $Part(S_0)$ que esté continuamente habilitada sin que, finalmente, se ejecute alguna acción de la clase. El criterio de progreso o viveza queda formalmente definido considerando los comportamientos equitativos del autómata S_0. Si un nodo x es señalado víctima porque tiene la identidad simulada mayor de todos los nodos candidatos de un ciclo, la acción $Abort_x$ permanecerá habilitada hasta que el interbloqueo se resuelva, es decir, hasta que finalmente se ejecute dicha acción.

En conclusión, los comportamientos equitativos del autómata S_0 formalizan tanto el criterio de seguridad como el de viveza. A pesar de ello, no se puede considerar que la especificación del problema sea directamente el autómata S_0. En el modelo formal de Autómatas de Entrada/Salida, una especificación es un tipo especial de módulo de planes en el que todas las acciones de su signatura son externas. Si se considera que la especificación es simplemente $Scheds(Fair(S_0))$, se están imponiendo más condiciones de progreso de las que realmente son precisas en este trabajo. Por tanto, para no incrementar la dificultad de la tarea que se está abordando, se va a proponer una especificación menos restrictiva en la que la propiedad de viveza sólo consista en conseguir que todos los interbloqueos que aparezcan en el sistema se resuelvan.

Haciendo uso del autómata S_0 es relativamente fácil definir todos los comportamientos válidos que constituyen la especificación del problema. En primer lugar, se debe definir un módulo de ejecuciones que se llamará en lo que sigue $E_0\prime$. Las acciones y los estados del módulo $E_0\prime$ coinciden con las acciones y los estados del autómata S_0. Sin embargo, las ejecuciones de $E_0\prime$ son aquellas ejecuciones de S_0 que se consideran correctas. Para determinar cuáles de las ejecuciones de S_0 son correctas es necesario formular con precisión el requisito que deben verificar. En la definición de ese requisito, a partir de ahora condición de progreso, están involucrados dos nuevos conjuntos:

- El conjunto que incluye todos los estados de S_0 en los que existe un ciclo $\mathcal{C}$ en el sistema, denominado $Interbloqueo_0(\mathcal{C})$. La definición formal de este conjunto se escribe como:

  $Interbloqueo_0(\mathcal{C}) = \{s \in states(S_0) \text{ tal que } \mathcal{C} \in cycles(s)\}$

- El conjunto formado por todas las acciones de S_0 cuyo efecto es el aborto de un nodo de un ciclo $\mathcal{C}$. Dicho conjunto se denota $Resolución_0(\mathcal{C})$ y su expresión formal es:

$$Resolución_0(\mathcal{C}) = \{Abort_i\colon i \in nodes(\mathcal{C})\}$$

Una vez que han sido definidos ambos conjuntos, la condición de progreso, $Prog_0$ se puede expresar de la siguiente manera:

$Prog_0 \equiv \forall\ \mathcal{C} \in cycles(\mathcal{N})$ se verifica que $Interbloqueo_0(\mathcal{C}) \hookrightarrow Resolución_0(\mathcal{C})$

El predicado de la condición de progreso asegura que, para todo ciclo que pudiera aparecer en una ejecución de S_0, siempre se dará alguna acción que rompa dicho ciclo tras un estado en el que exista ese ciclo. Dicho de otra manera, cualquier ciclo que pudiera aparecer en el sistema será finalmente resuelto.

En resumen, el módulo de ejecuciones de S_0, $E_0\prime$, contiene aquellas ejecuciones en las que todos los interbloqueos se resuelven. Formalmente $E_0\prime$ se define como sigue:

- $states(E_0\prime) = states(S_0)$
- $sig(E_0\prime) = sig(S_0)$
- $execs(E_0\prime) = \{\alpha \in execs(S_0)\colon \forall\ \mathcal{C} \in cycles(\mathcal{N})$ se verifica que $Interbloqueo_0(\mathcal{C}) \hookrightarrow Resolución_0(\mathcal{C})\}$

Con esta última definición, se aborda fácilmente la tarea de expresar $E_0\prime$ como el módulo de ejecuciones de S_0 cuyas ejecuciones cumplen $Prog_0$, o lo que es lo mismo, $execs(E_0\prime) = \{\alpha \in execs(S_0)\colon \alpha$ verifica $Prog_0\}$. En la siguiente sección, se va a proporcionar la especificación del problema de detección y resolución de interbloqueos, E_0, a partir del módulo de ejecuciones $E_0\prime$.

3.3.4. Especificación formal del problema

De acuerdo con el modelo formal de Autómatas de Entrada/Salida, la especificación de un problema consiste en una signatura externa de acciones y un módulo de planes que contenga todos los planes posibles y correctos [117]. Seguidamente, se

muestra la especificación formal del problema de resolución distribuida de interbloqueos en sistemas de petición de único recurso.

Definición 3.3.1. Un algoritmo distribuido para la resolución de interbloqueos en un sistema distribuido con un gestor que funciona bajo el modelo *SR* es correcto si y sólo si se puede formalizar en un objeto (autómata o módulo) que verifique la especificación que define el modelo de planes trivial de E_0, $Scheds(E_0\prime)$.

Por definición [114], $Scheds(E_0\prime)$ es el módulo de planes trivial definido por las acciones de $E_0\prime$ y los planes de todas las ejecuciones de $E_0\prime$:

- $sig(E_0) = sig(Scheds(E_0\prime)) = sig(E_0\prime)$
- $scheds(E_0) = scheds(Scheds(E_0\prime)) = scheds(E_0\prime) = \{\alpha \in execs(E_0\prime) \colon \alpha \mid ext(E_0\prime)\}$

Dado que la especificación E_0 es el módulo trivial de planes de un módulo de ejecuciones, $E_0\prime$, del autómata S_0, todo comportamiento de E_0 es también un comportamiento de S_0. Esto implica que todo comportamiento de E_0 verifica el criterio de viveza o progreso que se desea satisfacer.

Capítulo 4

Un algoritmo distribuido para la detección y resolución de interbloqueos

El algoritmo de detección de interbloqueos que se presenta en esta tesis trata de mantener las características de complejidad del algoritmo de elección de líder en [113], sin incurrir en los sobrecostes que supone la adaptación propuesta en [112],al problema de detección y resolución del interbloqueo. Para lograr este objetivo se utiliza la idea avanzada por Kshemkalyani en [56] de dotar al algoritmo de memoria. Con este fin, los mensajes que intercambian los nodos del sistema registran los caminos que están recorriendo. De este modo, los nodos, al almacenar esta información, *recuerdan* los caminos seguidos por los mensajes que han recibido. Esa memoria permite que los cambios en las relaciones de espera del sistema no hagan necesario recoger de nuevo la información que ha sido recopilada en instancias anteriores del algoritmo. Obviamente, esto obliga a controlar estrictamente qué mensajes recibe cada nodo y cuándo puede recibirlos. Dejando aparte esta tarea de control, el algoritmo sigue siendo el algoritmo $\mathcal{O}(n)$ de elección de líder de [113]. Cuando un nodo sospecha de la existencia de un interbloqueo, envía mensajes tratando de averiguar si es cierta esa situación y si el encargado de resolverla es él mismo. En caso de que sólo un nodo haya

iniciado dicha investigación y exista realmente un interbloqueo, el iniciador recibirá su propio mensaje y podrá resolver. Si hay más nodos candidatos a detectar y resolver el interbloqueo, será necesario que se comuniquen entre ellos para decidir cuál de ellos es finalmente el responsable de romper el interbloqueo.

En casos dinámicos es posible que el nodo creador de una instancia del algoritmo deje de formar parte del interbloqueo mientras su instancia persista en la *memoria* del algoritmo. Para evitar las disfunciones que este hecho podría ocasionar en el curso de la detección del interbloqueo, el algoritmo dispone de diversas estrategias: el uso de *identidades simuladas*, la redirección de mensajes y los mecanismos de retardo en la recepción de mensajes. Las identidades simuladas permiten que toda instancia "viva" del algoritmo tenga un nodo que simule ser su creador entre los nodos interbloqueados. Incluyendo diferentes tipos de redirección de mensajes, se consigue que los mensajes dirigidos a nodos que ya no forman parte del interbloqueo alcancen a sus representantes en el mismo. Por último, los mecanismos de retardo impiden que un nodo procese información hasta que pueda asegurarse de que dicha información sea consistente.

4.1. Descripción intuitiva del algoritmo

Para comprender de manera sencilla el funcionamiento del algoritmo es recomendable diferenciar entre su comportamiento básico estático y su modo de operación en entornos cambiantes. Siguiendo el mismo proceso que en el diseño real del algoritmo, se abordan en primer lugar los escenarios estáticos. La detección y resolución de un interbloqueo en este tipo de configuraciones, al igual que ocurre en la elección de líder para redes completas de [113], se compone de cuatro fases: iniciación, reconocimiento, colaboración y negociación/transferencia de información.

Tras una primera fase de iniciación en la que algunos nodos bloqueados espontáneamente se convierten en candidatos a detectores de un posible interbloqueo, se procede al reconocimiento de esos candidatos entre sí. Como en un interbloqueo las relaciones de espera conforman un ciclo, los candidatos lanzan mensajes de tipo ALG a sus predecesores con el objetivo de localizar exclusivamente al candidato que les precede. Los nodos que no son candidatos colaboran, en esta fase de búsqueda de un candidato predecesor, retransmitiendo el mensaje que le llega al siguiente nodo predecesor. Cuando estos mensajes de exploración alcanzan finalmente un nodo candidato, se desarrolla una fase de negociación, basada en la comparación de los identificadores. De todos los candidatos que existen en el ciclo, se elegirá al de mayor identificador como víctima. Se llama víctima a aquel nodo candidato que se elige para romper el ciclo de esperas y, por tanto, resolver el interbloqueo.

La comparación de los identificadores de los candidatos se realiza a dos niveles. El primer nivel se corresponde con la comparación directa de dos candidatos que se han puesto en contacto al explorar el ciclo de esperas y conlleva las siguientes operaciones:

- si un candidato recibe el mensaje ALG de un candidato de identificador mayor, éste almacena esa identidad.
- si un candidato recibe el mensaje ALG de un candidato menor, éste le replica con un mensaje directo de tipo AVS con su identificador para indicarle que, como es menor, no va a ser elegido como víctima y, por tanto, su instancia tendrá que ser suspendida.

Aplicando este criterio y sea cual sea la ubicación de los candidatos en el ciclo, se puede asegurar que al menos hay un candidato que guarda el identificador de otro candidato, esto es, el candidato predecesor del candidato que se convertirá en víctima. Del mismo modo, se garantiza que como mínimo en el ciclo surge un mensaje directo, el del candidato con mayor identificador (la futura víctima) al nodo candidato que le

sucede en el ciclo. Si el nodo candidato lee en el mensaje ALG su propia identidad, el nodo es el único candidato del ciclo y, por tanto, la víctima que desencadenará el fin del interbloqueo. En el caso de que en el ciclo hubiera solamente dos candidatos, después del intercambio de mensajes descrito anteriormente, se formaría un nuevo mensaje AVS del nodo que abandona la candidatura a víctima con destino al otro candidato. Cuando el candidato, el único que mantiene en su instancia del algoritmo activa, recibe este último mensaje, se ha logrado la detección del interbloqueo.

El segundo nivel de comparación se establece entre tres candidatos relacionados transitivamente. En esta configuración se asume que el candidato menor de los tres ha conseguido almacenar el identificador de su candidato sucesor y conoce el identificador de su candidato predecesor a través de un mensaje directo. Con esta información el candidato menor asume que deja de participar de la elección de víctima y opta por:

- enviar un mensaje directo de tipo $AVSRSP$ al candidato sucesor, si el identificador de su candidato sucesor es mayor que el de su candidato predecesor. Este tipo de mensaje tiene el mismo sentido que los mensajes iniciales de búsqueda de un candidato predecesor (en contra del sentido de las esperas del ciclo) y, además, su recepción conlleva el mismo efecto que éstos, o sea, almacenar la identidad del mayor candidato sucesor.

- enviar un mensaje directo de tipo AVS al candidato predecesor, si el identificador de su candidato sucesor es menor que el de su candidato predecesor. Este mensaje directo equivale al de réplica entre dos candidatos cuando el candidato predecesor posee un identificador mayor. En consecuencia, este mensaje circula a favor del sentido de las esperas del ciclo y el nodo que lo envía se retira de la detección del interbloqueo.

A pesar de que estos dos tipos de mensajes directos entre candidatos se diferencian en el sentido de propagación, a favor o en contra de las relaciones de espera del

ciclo, en ambos casos el sentido se elige siempre atendiendo a una norma común: el candidato menor debe emitir un mensaje con información del candidato mayor hacia el candidato de identificador intermedio. Además, los nodos de destino, en cualquiera de los dos casos, son candidatos que posteriormente tendrán que cancelar su instancia del algoritmo.

Sabiendo que la fase de negociación entre candidatos va seguida de un proceso de transferencia de información en el que se guarda el identificador del candidato mayor, se puede intuir que la selección de la víctima está asegurada. El número de candidatos va disminuyendo de uno en uno cada vez que se realiza la comparación expresa de tres candidatos. Al quedar dos candidatos, sólo tendrá sentido el envío de un mensaje directo tipo *AVS* del candidato menor al candidato mayor, del mismo modo que si el ciclo hubiera contado con dos candidatos desde el principio. Como la información que se transfiere en este mensaje coincide con el identificador del candidato destino, se concluye que existe un ciclo de esperas (un interbloqueo).

Por último, hay que comentar que se ha convenido que el mismo candidato que detecta el interbloqueo sea el encargado de resolverlo (acción *Abort* del algoritmo). Para ello, como efecto de la recepción del mensaje final de detección, el candidato mayor del ciclo se convierte en la víctima que, al desvincularse del mismo, hará que desaparezca el interbloqueo. Una víctima se desvincula de un ciclo rompiendo las relaciones de espera que lo unen con sus predecesores (nodos bloqueados por él) y la relación de espera que lo une con su sucesor (nodo por el que quedó bloqueado).

Todo lo explicado anteriormente se corresponde con la detección y resolución de un interbloqueo en el que las relaciones de espera que lo componen no se han visto modificadas desde el momento en que se iniciaron y hasta que se formó el ciclo. A continuación, se va a describir el funcionamiento del algoritmo en escenarios que no permanecen fijos porque las relaciones de espera (arcos en el grafo) se modifican a la vez que se ejecutan las instancias del algoritmo. Al igual que en el proceso de diseño del

algoritmo llevado a cabo para esta tesis, después de afrontar los escenarios estáticos se contemplan los escenarios dinámicos que pueden considerarse más realistas que los anteriores. Cuando se admite que las relaciones de espera que forman el ciclo final se han incorporado a él después de haber participado en la ejecución de una instancia del algoritmo que se haya interrumpido o concluido, se pueden adoptar dos formas antagónicas de afrontar la detección de un interbloqueo. Una opción sería iniciar la ejecución del algoritmo como si se tratase de un nuevo escenario estático [112] y otra, reutilizar la información que se recopiló en la ejecución de instancias suspendidas o acabadas.

El algoritmo presentado en este trabajo desarrolla la segunda opción e incluye una serie de mecanismos para aprovechar la información útil que quedó almacenada en la ejecución de instancias que ya no tienen vigencia. La adaptación de las tareas de detección y resolución de interbloqueos a estos escenarios cambiantes se ha efectuado teniendo en cuenta dos ideas claves. La primera idea consiste en que, siempre y cuando la aparición y desaparición de las relaciones entre los nodos provoque cambios en el grafo del sistema, se reutilizará toda la información útil que se haya podido recopilar hasta el momento de la modificación. La segunda idea que regirá el proceso de dinamización del algoritmo es que, mientras la información salvaguardada se corrija, redireccione y/o comunique a los nodos que lo precisen, las instancias del algoritmo no avanzarán en las operaciones propias de la detección y sólo podrán registrar los cambios existentes en el medio.

Como es de suponer, dotar de carácter dinámico al algoritmo descrito para situaciones estáticas, no resulta sencillo. Por eso, en este apartado inicial del capítulo sólo se pretende ir presentando los mecanismos principales que "dinamizan" el algoritmo, sin entrar en detalle de cómo se han desarrollado. Asimismo, se dejarán entrever las necesidades que surgen en las distintas evoluciones del sistema. En lo que resta de

capítulo se explicarán pormenorizadamente todas las acciones del algoritmo y su relevancia para detectar y resolver interbloqueos conforme cambian las relaciones de espera entre los nodos del sistema.

En primer lugar, se va a plantear la formación más sencilla de un ciclo a partir de una sucesión de esperas. Si un nodo candidato difunde su identidad entre todos los nodos que le preceden sin encontrar otro candidato, con posterioridad se rompen las relaciones de espera que lo unen con sus predecesores más próximos y, finalmente, se forma un ciclo con los predecesores que se mantienen de la configuración inicial, el algoritmo deberá ser capaz de detectar el interbloqueo aunque el candidato ya no esté presente. Para ello, sabiendo que la identidad del candidato desaparecido ha quedado almacenado en todos los nodos del ciclo, se idea un método por el cual el nodo del ciclo que estuvo más cerca del candidato hará las veces de éste en la detección del interbloqueo. Este cambio aparente de la identidad de un nodo parece sencillo pero conlleva incorporar el concepto de *identidad simulada*. Los nodos candidatos se seguirán caracterizando por su identificador pero, en la fase de negociación previa a la selección de la víctima, las comparaciones entre candidatos se realizarán teniendo en cuenta la *identidad simulada*.

Por otra parte, resulta evidente que hasta que el nodo del ciclo simule la presencia del candidato desaparecido, esto es, adquiera su nueva identidad, la detección debe posponerse. Para que se respete esta condición, el nodo va a adoptar un nuevo estado (*unknown*) con el que pondrá de manifiesto que desconoce y está a la espera de recibir la identidad simulada que tiene que asumir. Cuando el nodo adquiera esa identidad finalmente, el estado del nodo volverá a ser el que permite desarrollar con normalidad las tareas de detección (*known*). Mientras tanto, las acciones de intercambio de algunos mensajes del algoritmo entre candidatos quedan paralizadas.

Otro aspecto relacionado con la necesidad de que un nodo cambie su identidad es la manera en la que le va a llegar su nueva identidad simulada. La solución adoptada

requiere que, cuando un nodo haya procesado mensajes antes de empezar a romper las esperas con sus predecesores, la identidad (simulada) de ese nodo se envíe a sus predecesores en un mensaje especial (mensaje de tipo *INF*). Así pues, en la situación que se está describiendo, la identidad del nodo candidato se incluye en un mensaje *INF* con el mismo sentido que el mensaje *ALG* inicial que sirvió para difundir su identificador al predecesor del que se ha desligado. Un nuevo mensaje de este tipo será enviado por cada uno de los predecesores que retransmitieron la identidad del candidato y que han roto la espera que les une al ciclo a detectar.

Antes de continuar con el escenario dinámico que se ha planteado hay que destacar las características que hacen de la identidad simulada tenga una gran relevancia en este algoritmo. Se ha asumido en el modelo del sistema que los identificadores de los nodos deben ser únicos. En consecuencia, es lógico pensar que las identidades simuladas, construidas a partir de los identificadores de los nodos, también deberían contar con la propiedad de unicidad. Por otro lado, el modelo de petición de recursos que se ha impuesto en este trabajo es el de único recurso. Este tipo de petición de recursos permite que aparezcan relaciones de espera de varios procesos por un mismo recurso o varios arcos apuntando a un mismo nodo en una representación gráfica. Considerando una configuración como la que se acaba de mencionar, un nodo candidato podría difundir su identificador por varios caminos y, al desvincularse de sus predecesores, éstos deberían heredar su identidad. El éxito de la detección de un ciclo en el que sus nodos candidatos han adquirido la identidad de un mismo candidato estriba en que, pese a tener el mismo identificador y tiempo de activación, posean una componente que las distinga. Esa parte de la identidad simulada que hace que todas las identidades simuladas sean distintas es la que hace referencia a los diferentes caminos por los que el identificador de un candidato se propagó inicialmente. Conforme la identidad simulada va pasando de predecesor en predecesor, se añade en ésta una lista de marcas, una por cada uno de los reenvíos que va experimentando. Esta componente

de la identidad simulada equivale a señalar las esperas rotas que se van superando en su propagación.

Retomando el caso dinámico que se está analizando, falta indicar que, al recibir y procesar el mensaje *INF*, el nodo del ciclo que finalmente adquiere la identidad simulada, cambia su estado a (*known*) para reanudar las acciones que permitan progresar en la detección. La rececepción de un mensaje tipo *ALG*, al igual que sucede en los casos estáticos con sólo un nodo candidato, hace que el "nuevo" nodo candidato se convierta en víctima y resuelva el interbloqueo existente.

Una vez explicada la evolución dinámica más simple posible, corresponde ahora plantear otras formaciones en las que resulte un ciclo con más de un nodo candidato. Los nodos candidatos que contenga el ciclo han podido llegar a ese estado después de un proceso de borrado de esperas y la herencia de una identidad simulada o ser candidatos desde que se bloquearon e iniciaron una instancia del algoritmo. Sea cual sea la forma de haberse convertido en candidato, la manera de elegir cuál de ellos se convertirá en víctima será similar a la de los casos estáticos. La fase de negociación del algoritmo consistirá en que los candidatos finales del ciclo intercambien mensajes y comparen sus identidades simuladas. Tras la comparación de las identidades simuladas de los candidatos, del mismo modo que en el ciclo estático, el candidato que conoce las identidades de un candidato que le precede y de otro que le sucede puede mandar un mensaje de tipo *AVS* o un mensaje de tipo *AVSRSP*. Si su candidato sucesor es mayor que su predecesor entonces, se envía la información en un mensaje *AVSRSP* al candidato predecesor. Por el contrario, si su candidato sucesor es menor que su predecesor, se pasa toda la información recopilada en un mensaje *AVS* al candidato sucesor. Es justo en este momento en el que se envían los mensajes, cuando se puede intuir la necesidad de incorporar un nuevo mecanismo en el algoritmo. Si el nodo al que se le manda un mensaje *AVS* ya no forma parte del ciclo a detectar, el mensaje *AVS* se mandará fuera del ciclo. Más concretamente, el mensaje *AVS* se dirige al

nodo al que hace alusión el identificador de la identidad simulada que ha heredado un candidato del ciclo final. Para que la detección pueda proseguir en esta situación, se requiere que el mensaje *AVS* retroceda hasta el nodo candidato que lo representa en el ciclo final. Este mecanismo, al que se ha denominado *retroceso de mensajes AVS*, precisa conocer el camino de nodos que debe deshacer hasta alcanzar un nodo que pertenezca al ciclo. En consecuencia, todos los mensajes intercambiados entre nodos deben incluir las rutas, secuencias de nodos que se dan por conocidas al generarse un mensaje.

Advertido el motivo por el que el algoritmo debe contar con memoria, hay que aclarar que todos mensajes que fluyen en el sistema a excepción de los mensajes de tipo *INF* recogen básicamente la secuencia de nodos que describen las relaciones de esperas que van desde el nodo emisor hasta el nodo destino del mensaje. Si el nodo emisor conoce la secuencia de nodos que lo une a su candidato predecesor o a su candidato sucesor también la incorpora ordenadamente en el mensaje. En el caso dinámico es obvio que las secuencias de nodos que quedan registradas en los mensajes tendrán una parte correcta, correspondiente a esperas reales en el sistema, y otra parte incorrecta, que procede de esperas que ya no existen en el sistema. Eliminando de los mensajes directos la parte de la ruta de nodos que no se corresponde con las esperas reales y comparando los candidatos entre sí con la identidad simulada, se logra que el mensaje final que sirve para detectar incluya una secuencia ciclíca de nodos y el nodo que se convierta en víctima sea el candidato de mayor identidad simulada.

En relación con el retroceso de mensajes *AVS*, surge también la necesidad de transformar rutas almacenadas procedentes de este tipo de mensajes. En caso de que un nodo candidato haya adquirido una nueva identidad simulada, mayor que la que poseía, la información que ha podido quedar almacenada no será consistente porque la identidad de referencia para la fase de negociación y transferencia es mayor. La forma ideada de aprovechar la ruta almacenada en esta situación será incluir esta

información en un mensaje antes de eliminarla. Se convertirá lo que fue un mensaje de tipo *AVS* en un mensaje *AVSRSP*, avisando al candidato que se creía con mayor identidad simulada de que existe otro candidato que opera con una identidad superior. Este mecanismo se llamará en lo que sigue *reconversión de mensajes AVSRSP*.

Aplicando estas consideraciones a la transferencia de información entre candidatos de un ciclo en entornos dinámicos, se consigue asimilar la detección de un interbloqueo en escenarios estáticos con dos o más candidatos a la de escenarios dinámicos. Las instancias del algoritmo van siendo finalizadas de una en una, conforme se suceden las negociaciones entre candidatos.

Aunque existen más situaciones particulares derivadas del dinamismo que hay que tener en cuenta, la descripción realizada puede servir como punto de partida para comprender el funcionamiento global del algoritmo. Mecanismos concretos como: el que sirve para evitar que se emitan mensajes que no aportan información, el que elimina mensajes del canal de comunicación que no podrán ser tratados porque contienen información obsoleta o el mecanismo que transforma información almacenada que deja de ser coherente y la ubica en otros nodos del sistema donde aún puede ser utilizada; son ejemplos de operaciones que están integradas en el algoritmo pero no reflejan los fundamentos de la detección y resolución de un interbloqueo.

4.2. Descripción formal del algoritmo

En esta sección se define formalmente el algoritmo propuesto como solución al problema de detección y resolución de interbloqueos. Al igual que para formalizar el gestor del sistema, G, se va a hacer uso del modelo de autómatas de Entrada/Salida. Este autómata que se llamará en lo que sigue, A, debe incluir las acciones necesarias tanto para la detección de interbloqueos como para su posterior resolución.

4.2.1. Definición del autómata del algoritmo, *A*

Estado de A:

El algoritmo necesita almacenar y gestionar diversas informaciones para lograr su objetivo. Por cada nodo i, se dispondrá de una serie de variables que reflejen el conocimiento de:

- su estado durante la ejecución del algoritmo, $status.alg_i$ y $status.id_i$,
- sus características temporales, ta_i y t_unk_i,
- su identidad simulada en la ejecución del algoritmo, sim_id_i,
- la identidad del nodo candidato por el que espera, $cand\_succ_i$,
- las identidades de los nodos que lo esperan, set_Pred_i,
- las identidades de los nodos con los que ha entablado comunicación, $set\_PredToInf_i$,
- el número de predecesores de los que se ha desligado, $cont_i$,
- el envió de algunos mensajes, $nofirstAVS_i$ e inf_need_i,
- las características del sistema obtenido por los mensajes que ha intercambiado, st_alg_i, st_avsrsp_i y $set\_st\_avs_i$.

El estado del autómata A no sólo viene definido por los valores de las variables asociadas a cada nodo i del sistema. La definición del estado de A se completa con la información de los cuatro tipos de mensajes que se intercambian: mensajes ALG, mensajes AVS, mensajes $AVSRSP$ y mensajes INF. Estos mensajes pueden ocupar los canales de comunicación establecidos entre cualquiera par de nodos del sistema. Cada uno de estos mensajes comienza con un campo, *type*, que lo identifica según su funcionalidad. A continuación, se describe la estructura y características de los diferentes mensajes:

- Un mensaje ALG se compone de cuatro componentes ($type = ALG$, t, sid, $path$). Después de la identificación del mensaje (campo $type$) aparece: la marca temporal que se espera tenga el nodo destino del mensaje (t). Esta marca ayuda a chequear la validez del mensaje cuando se recibe. A continuación aparecen la identidad simulada del emisor del mensaje (sid) y la ruta o secuencia de nodos por la que se ha retransmitido el mensaje ($path$) desde que se inició la instancia que lo generó. Este tipo de mensaje lo mandan los nodos que inician una instancia del algoritmo a todos sus predecesores directos. A su vez, éstos lo reenvían a sus propios predecesores, manteniendo el campo sid y registrando en la ruta, $path$, su identidad. El alcance de las retransmisiones de un mensaje ALG viene determinado por la localización de otro nodo iniciador del algoritmo. La trayectoria de los mensajes ALG coincide en dirección con las relaciones de espera de los nodos del sistema pero su sentido es contrario al que éstas señalan.

- Un mensaje $AVSRSP$ está constituido por los mismos cuatro campos ($type = AVSRSP$, t, sid, $path$). La información de este tipo de mensaje se organiza de forma similar a la de los mensajes ALG. En el campo t se guarda el tiempo que se espera tenga el nodo al que va dirigido el mensaje, la componente sid contiene la identidad de un candidato, el mayor de los sucesores que se conocen después del intercambio de mensajes entre varios candidatos y el campo $path$ incluye la secuencia de nodos que va desde el nodo destino del mensaje hasta el nodo registrado en el campo sid. Los mensajes $AVSRSP$, a diferencia de los mensajes ALG, no siguen la red de predecesores para su propagación. En consecuencia, se dice que son mensajes directos que ponen en contacto a dos candidatos cuando ya se conocen las relaciones de espera transitiva que existen entre ellos.

- Un mensaje AVS se estructura en cinco campos ($type = AVS$, sid, t, $path$, fwd). Los cuatro primeros campos sirven para almacenar datos semejantes a los que

se han comentado en los tipos anteriores de mensajes pero el último campo es propio de los mensajes *AVS*. El campo *sid* contiene en este caso la identidad del candidato predecesor mayor que se conoce tras la negociación de varios candidatos, el tiempo del nodo al que se manda el mensaje se guarda en t y para almacenar la ruta de nodos que va desde el nodo almacenado en *sid* hasta el nodo destino del mensaje, se cuenta con el campo *path*. El último campo de un mensaje *AVS*, *fwd*, es un booleano que permite distinguir si el mensaje va a deshacer el camino que ya recorrió ($fwd = true$), o el mensaje va a alcanzar un nuevo destino ($fwd = false$). En el primer caso, se usa la ruta de un mensaje *AVS* almacenado para generar la nueva ruta del mensaje en retroceso. En cambio, en el segundo caso, el mensaje se ha formado combinando un mensaje *AVS* y *AVSRSP* previamente almacenados. Por otra parte, el destino de un mensaje *AVS* con *fwd* a *true* es el nodo que aparece en la ruta como su predecesor. Sin embargo, para un *AVS* con *fwd* a *false* el destino viene dado por el último nodo de la ruta que hay almacenada en *path*.

- Un mensaje *INF* está formado por tres componentes ($type = INF$, sid, t). Tras el campo que identifica el tipo del mensaje, se encuentra el campo *sid* que contiene el valor de una identidad simulada. Se trata de la identidad simulada que el nodo emisor del mensaje *INF* quiere propagar a su predecesor, la componente t del mensaje está reservada para el tiempo de activación de ese nodo destino. La recepción de un mensaje *INF* está regida por esta marca temporal y el estado del nodo receptor.

Antes de formalizar el conjunto de variables y mensajes que describen el estado del autómata del algoritmo A es necesario establecer las siguientes definiciones:

Definición 4.2.1. Se denomina T al conjunto de posibles ternas (*identificador de un nodo*, *tiempo de activación nodo*, *cadena de números naturales*) o expresado matemáticamente, $T = \{(n, t, \alpha): n \in \mathcal{N} \wedge t \in \mathbb{N} \wedge \alpha \in \mathbb{N}^*\}$.

Definición 4.2.2. Se denota como M_{ALG} al conjunto de mensajes ALG que pueden surgir en la ejecución del algoritmo tal que $M_{ALG} = \{(ALG, t, sid, path): t \in \mathbb{N} \wedge sid \in T \wedge path \in P^+\}$.

Definición 4.2.3. El conjunto de posibles mensajes AVS que surgen en la ejecución del algoritmo se define como $M_{AVS} = \{(AVS, sid, t, path, fwd): sid \in T \wedge t \in \mathbb{N} \wedge path \in P^+ \wedge fwd \in \{true, false\}\}$.

Definición 4.2.4. Los posibles mensajes $AVSRSP$ que pueden surgir en la ejecución del algoritmo forman un conjunto que queda definido como $M_{AVSRSP} = \{(AVSRSP, t, sid, path): t \in \mathbb{N} \wedge sid \in T \wedge path \in P^+\}$.

Definición 4.2.5. M_{INF} es el conjunto de mensajes INF que pueden aparecer en la ejecución del algoritmo tal que $M_{INF} = \{(INF, sid, t): sid \in T \wedge t \in \mathbb{N}\}$.

Definición 4.2.6. Se designa por M al conjunto de todos los posibles mensajes del algoritmo que pueden circular en el sistema. Considerando los diferentes tipos de mensajes que existen, M se puede definir como la unión de los conjuntos: M_{ALG}, M_{AVSRSP}, M_{AVS} y M_{INF}, esto es, $M = M_{ALG} \cup M_{AVSRSP} \cup M_{AVS} \cup M_{INF}$.

Descripción de las variables del algoritmo asociadas a cada nodo del sistema:

La variable $\boldsymbol{status.alg_i}$ permite distinguir el estado en el que se encuentra el nodo i durante la ejecución del algoritmo. Se han considerado seis estados posibles: *active*, *blocked*, *candidate*, *dummy*, *victim* y *aborted*. Si $status.alg_i = active$, valor inicial de la variable, el nodo i no está a la espera de ningún nodo. Un nodo cuyo valor de *status.alg* sea *blocked* es un nodo que ha iniciado una relación de espera por otro nodo. Por otra parte, sólo los nodos en estado *blocked* pueden iniciar el algoritmo y, en consecuencia, pasar a estado *candidate*. Todo nodo candidato transmite un mensaje ALG en búsqueda de otros iniciadores del algoritmo. Cuando este mensaje llega a un nodo *blocked*, éste pasa a estado *dummy*, con lo que queda excluido del proceso de detección de interbloqueos. De entre todos los nodos candidatos presentes en un interbloqueo, sólo uno cambia a estado *victim*. Este estado implica que se ha detectado un interbloqueo y que la resolución del mismo conlleva que el nodo en estado *victim* aborte. Finalmente, el estado *aborted* es el valor de $status.alg_i$ cuando el nodo i rompe la situación de interbloqueo detectada.

Como complemento a la variable $status.alg_i$ se ha definido otra variable de estado llamada $\boldsymbol{status.id_i}$. Esta nueva variable no indica el papel que el nodo desempeña en la ejecución del algoritmo sino que aporta información sobre el conocimiento de su identidad simulada. Si $status.id_i = known$, el nodo i dispone de la identidad que el algoritmo precisa para sus operaciones. En cambio, si el valor de *status.id* para el nodo i es *unknown*, el nodo está a la espera de conseguir una nueva identidad simulada que utilizará en el proceso de selección de la víctima.

La figura 4.1 muestra las transiciones que pueden darse entre los valores de la variable $status.alg_i$ combinados con los valores de la variable $status.id_i$. En la figura se han utilizado flechas de distintos colores para representar las transiciones de estados propias de la ejecución del algoritmo en una cofiguración estática (flechas negras) y las

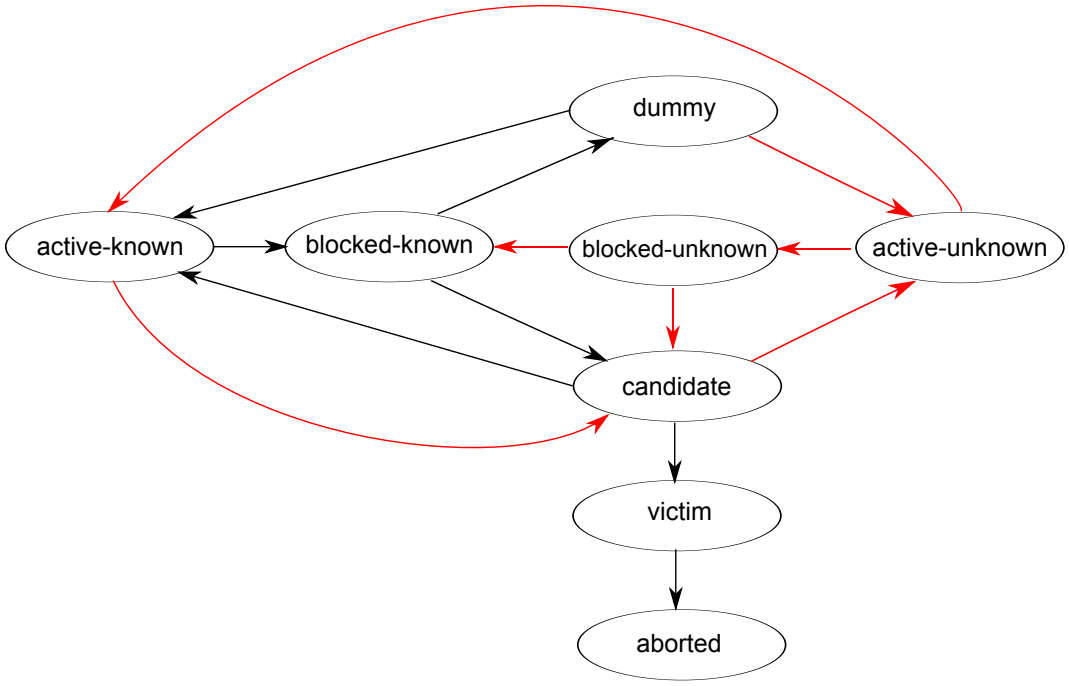

Figura 4.1: Transiciones posibles de los valores de las variables *status.alg* y *status.id* en el autómata A

transiciones de estados que implican una evolución del sistema en la que, previamente a la aparición y el borrado de esperas, se ha realizado un traspaso de información con otros nodos (flechas rojas).

En el momento de la puesta en marcha del algoritmo, el nodo i está activo, $status.alg_i = active$, y conoce la identidad con la que participa en el sistema, $status.id_i = known$. Posteriormente, el nodo podrá bloquearse pasando $status.alg_i$ a ser *blocked* y manteniendo $status.id_i = known$. Cuando el nodo esté bloqueado, dado que podría formar parte de un interbloqueo, la copia local del algoritmo en el nodo i podría iniciar el proceso de detección. En ese caso, las variables del nodo pasan a valer $status.alg_i = candidate$ y $status.id_i = known$. Si el algoritmo detecta que el nodo i está interbloqueado, se señalará al nodo i como la víctima que, al desaparecer del sistema, podrá resolver la situación de interbloqueo, $status.alg_i = victim$ y $status.id_i$

= *known*. Una vez que el nodo i haya alcanzado el estado de víctima, tarde o temprano abortará, resultando $status.alg_i = aborted$ y $status.id_i = known$.

Si el proceso de detección alcanza un nodo bloqueado, con $status.id_i = known$, que no ha iniciado la ejecución de su copia del algoritmo, el nodo adoptará un nuevo estado ($status.alg_i = dummy$ y $status.id_i = known$) que lo identifica como un nodo que no podrá señalar una situación de interbloqueo aunque esté incluido en él.

Tanto si se trata de un nodo con $status.alg_i = dummy$ como con estado $status.alg_i = candidate$, el nodo puede llegar a desbloquearse, cancelando el proceso de detección que lo había analizado. En cualquiera de estos dos casos el nodo i vuelve a ser activo, $status.alg_i = active$. En un funcionamiento normal en el que el proceso de detección no ha avanzado e intercambiado información con otros nodos, el nodo se activa manteniendo la identidad de su participación en la detección, esto es, conoce su identidad y, por tanto, $status.id_i = known$.

A continuación se van a describir las transiciones de estado que en la figura 4.1 aparecen dibujadas mediante flechas rojas. Suponiendo que desaparece del grafo la espera que parte de un nodo i con $status.alg_i = \{dummy, candidate\}$ y que el proceso de detección del interbloqueo hubiera evolucionado con la participación del nodo que se activa, el nodo i debería quedar a la espera de conocer la identidad simulada con la que se incorporará en el proceso de detección con los nodos que no se han retirado de la detección iniciada o posteriormente se sumen al ella. El estado final que representa esa situación, fruto de un comportamiento dinámico del sistema, es $status.alg_i = active$ y $status.id_i = unknown$.

Mientras un nodo i con $status.alg_i = active$ y $status.id_i = unknown$ espera para conocer la identidad con la que proseguirá el proceso de detección del que todavía participa, éste puede bloquearse por un nuevo nodo. La ejecución de la transición que permite el nuevo bloqueo convierte su estado en $status.alg_i = blocked$ pero mantiene $status.id_i = unknown$.

El algoritmo también permite que un nodo con $status.id_i = unknown$ se reenganche al proceso de detección en cuanto reciba la información que precisa. Una vez que el nodo i conoce la identidad simulada con la que continuará participando en la detección del interbloqueo, pasa a tener $status.id_i = known$. Esta situación puede tener lugar siendo, o bien $status.alg_i = active$, o bien $status.alg_i = blocked$. Cuando el nodo i está activo y adquiere la identidad que empleará desde ese momento en el proceso de detección, su estado se mantiene como $status.alg_i = active$, pero a partir de ese instante $status.id_i = known$. Algo similar sucede en el caso de que, antes de adoptar una nueva identidad para continuar la ejecución del algoritmo, el estado del nodo i sea $status.alg_i = blocked$. Una vez que el nodo conozca la identidad que simula, su estado seguirá siendo $status.alg_i = blocked$ y se modificará únicamente $status.id_i$, pasando a valer $status.id_i = known$. Una transición distinta a la mencionada es posible si el nodo i en estado *blocked-unknown* mantiene esperas reales en el grafo con nodos predecesores con los que ya ha intercambiado información. En este caso, después de que el nodo i asuma una identidad para el proceso de detección de interbloqueo, el nodo i alcanzará el estado $status.alg_i = candidate$ con $status.id_i = known$.

Finalmente, hay que comentar el cambio de estado que se produce cuando un nodo i en estado *active-known* va a iniciar un nuevo bloqueo, pero queda constancia de que anteriormente estuvo bloqueado e intercambió información con nodos que aún siguen siendo sus predecesores. En esta situación, el nodo i se convierte directamente en un candidato, $status.alg_i = candidate$, para el proceso de detección y resolución que lleva a cabo el algoritmo. Por lo tanto, resulta posible la transición directa de $status.alg_i = active$ y $status.id_i = unknown$ a $status.alg_i = candidate$ y $status.id_i = known$ sin necesidad de convertirse previamente en un nodo con $status.alg_i = blocked$.

Además de un identificador único, un nodo i cuenta con una referencia temporal, $\boldsymbol{ta_i}$, que el algoritmo emplea para contar el número de veces que se ha bloqueado o activado. Inicialmente, $ta_i = 1$ y cada vez que el nodo i vuelve a ser activo, después

de haber estado bloqueado, el tiempo se incrementa en una unidad. Con el valor de esta variable temporal, el nodo i puede validar los mensajes que recibe. Si el valor del tiempo del nodo destino que aparece en el mensaje no coincide con el del nodo en el momento de la recepción, el mensaje se rechazará porque ha quedado obsoleto.

En el algoritmo se utiliza otra variable temporal, $\boldsymbol{t\_unk}_i$. El valor inicial de esta variable es -1 y en la ejecución del algoritmo puede contener cualquier número natural. Su uso viene determinado por la necesidad de recordar el instante de tiempo en el que el nodo i recibió información procedente de un nodo candidato a través de un mensaje ALG. Si el nodo i se ve inmerso en una modificación de las relaciones de espera que lo desvinculan de ese candidato es posible que requiera una nueva identidad simulada. En caso de que no precise una identidad simulada, la variable recuperará su valor inicial.

La variable $\boldsymbol{cont}_i$ es un contador que comienza en el valor 1 y está definido en el conjunto de los números naturales. Para que este contador pueda cambiar de valor es preciso que el nodo i haya intercambiado información con sus predecesores y rompa la relación de espera que le une a ellos. Por cada uno de los predecesores eliminados, el contador se incrementa en una unidad. La variable $cont_i$ no se modifica o vuelve al valor inicial en caso de que la relación de espera que se vaya a eliminar sea única.

En $\boldsymbol{sim\_id}_i$ se almacena la identidad simulada que el nodo i adopta conforme evoluciona la ejecución del algoritmo. Esta variable toma sus valores del conjunto T. Los tres campos que componen una identidad simulada se corresponden con el identificador de un nodo ($sim\_id_i.id$), el tiempo de activación de ese nodo ($sim\_id_i.ta$) y una cadena de números naturales ($sim\_id_i.nu$). La identidad simulada inicial del nodo i es (i, ta_i, ϵ). Si el nodo i no se ve involucrado en ninguna evolución dinámica, el valor inicial de sim_id_i no se modifica durante la ejecución del algoritmo. Además podría considerarse, en ese caso, que su identidad simulada equivale únicamente a los dos primeros campos de sim_id_i.

Una característica relevante de la variable sim_id_i es que no hay dos nodos en el sistema que tengan la misma identidad simulada. Cuando el nodo i debe propagar su identidad simulada a los predecesores con los que mantenía una relación de espera, se generan distintos valores para cada uno de ellos. Modificando el tercer campo de sim_id_i se consigue distinguir identidades simuladas que procedan del mismo nodo i, pero que se traspasan a diferentes predecesores. Para ello, en la cadena de números de sim_id_i se añade el valor del contador $cont_i$ como elemento final de la misma. Como se ha explicado anteriormente, este contador se va incrementando cada vez que un nodo rompe su relación de espera con uno de sus predecesores a los que tiene que hacer llegar si identidad. Por tanto, con este mecanismo de formación de identidades simuladas para propagar, se garantiza que nunca dos nodos adquirirán identidades simuladas iguales, independientemente del trayecto de propagación que se siga.

Para registrar la identidad simulada del candidato sucesor mayor que conoce el nodo i durante la ejecución del algoritmo, se utiliza la variable $\boldsymbol{cand\_succ_i}$. Como en ella se incluyen valores que representan identidades simuladas, $cand\_succ_i$ tiene la misma estructura que sim_id_i y está definida sobre el mismo conjunto T. Los nodos del sistema inicialmente no tienen ningún valor ($NULL$) en $cand\_succ_i$. Sin embargo, conforme los nodos candidatos intercambian mensajes entre sí, la variable $cand\_succ_i$ puede ir cambiando de valor.

Con el fin de conocer en todo momento los predecesores del nodo i, la variable $\boldsymbol{setPred_i}$ guarda sus identificadores y registros temporales. Según esto, los valores de $setPred_i$ son pares de datos que pertenecen al conjunto P. En la puesta en marcha del sistema, se asume que $setPred_i$ es un conjunto vacío. Cada vez que se completa una relación de espera por el nodo i, se incorpora en $setPred_i$ el identificador del nodo que queda bloqueado por i y el instante de tiempo en el que ese nodo se activó por última vez. Cuando la relación de espera con un predecesor de i se rompe, el conjunto de predecesores se actualiza eliminando de $setPred_i$ su información.

Del mismo modo que en la variable $setPred_i$, en $\boldsymbol{setPredToInf}_i$ también se introducen los identificadores y los tiempos asociados de los nodos predecesores del nodo i. Por eso, los posibles valores de esta variable, que inicialmente es el conjunto vacío, también son pares de datos que pertenecen a P. Sin embargo, para que los datos de un nodo predecesor de i se añadan a $setPredToInf_i$ es imprescindible que el nodo i le haya enviado un mensaje de tipo ALG a ese predecesor. Por otra parte, la información de un predecesor de i no desaparecerá de $setPredToInf_i$, si en la activación del nodo i éste no quedó a la espera de una nueva identidad simulada. Esto quiere decir que es en las evoluciones dinámicas del sistema cuando se advierte la diferencia entre las variables $setPred_i$ y $setPredToInf_i$.

La variable $\boldsymbol{nofirstAVS}_i$ se ha ideado para indicar que un nodo i, siendo candidato, no ha enviado todavía un mensaje AVS en respuesta de un mensaje ALG de otro candidato de identidad simulada inferior. Inicialmente, el valor de esta variable para todos los nodos del sistema es *true*. Sólo los nodos candidatos que se ven en la situación mencionada cambian el valor de este booleano a *false* cuando mandan el mensaje AVS. Una vez que el nodo candidato ha señalado que ha enviado este primer mensaje AVS y mientras la instancia del algoritmo que inició esté activa, su variable $nofirstAVS$ no vuelve a cambiar de valor.

Otra variable booleana que utiliza el algoritmo es $\boldsymbol{inf\_need}_i$. Su valor *true* sirve como señal de que un nodo candidato ha generado un mensaje AVS o $AVSRSP$ a partir de la información que tenía almacenada. Si este candidato se activara, quedaría esperando una nueva identidad simulada porque ha intervenido como candidato en el proceso de selección de una víctima que resuelve el interbloqueo.

Finalmente, por cada par de nodos del sistema se define un canal de comunicación $\boldsymbol{channel(i, j)}$ para que el envío de mensajes entre ellos pueda ser directo.

Descripción de las variables del algoritmo dedicadas al almacenamiento de los mensajes que recibe un nodo *i*:

Los mensajes *ALG* que recibe y procesa el nodo i se almacenan en la variable $\boldsymbol{st\_alg_i}$. Esta variable consta de dos campos: *sid* y *path*. En el primero de ellos, se guarda directamente el valor que aparece en el mensaje *ALG* y se corresponde con la identidad simulada de un candidato. Como la identidad simulada es una variable que toma sus valores del conjunto T, el campo *sid* de st_alg_i está definido en el mismo conjunto. Por otra parte, la componente *path* de st_alg_i alberga rutas o secuencias (nodo, tiempo) que se extraen del campo *path* del mensaje *ALG*. Así que, *path* debe contener información del tipo definido en el conjunto P^+.

Cuando el nodo i recibe un mensaje *AVSRSP* válido se almacena en $\boldsymbol{st\_avsrsp_i}$. Para ello, la variable st_avsrsp_i cuenta con los campos (t, *sid*, *path*) y se completan con los valores correspondientes a los mismos campos en el mensaje *AVSRSP*. También es posible completar esta variable con la información de un mensaje *ALG*. Si un nodo candidato recibe un mensaje *ALG* de otro candidato con mayor identidad simulada que él, almacena directamente el contenido de los campos t, *sid* y *path* del mensaje en los respectivos campos de st_avsrsp_i.

Por último, para almacenar los mensajes *AVS* que llegan al nodo i y son de interés en un proceso de detección de interbloqueo, se dispone de la variable $\boldsymbol{set\_st\_avs_i}$. A excepción del campo correspondiente al tipo de mensaje, el contenido del *AVS* se traspasa a los respectivos campos de la variable $set\_st\_avs_i$, (*sid*, t, *path*, *bool*). El almacenamiento de los datos es consistente porque, al igual que en el mensaje *AVS*, el campo *sid* está definido en el conjunto T, t pertenece a los números naturales, las rutas que aparecen en *path* son elementos del conjunto P^+ y el campo booleano (equivalente al campo *fwd* del mensaje) sólo puede tomar los valores *true* o *false*. Hay situaciones en las que el valor que se guarda en el campo *bool* es el contrario al que aparece en el campo *fwd* del mensaje. Por ejemplo, algunos mensajes *AVS* que se

generan en el mecanismo de retroceso, $fwd = true$, se almacenan en $set\_st\_avs_i$ con $bool = false$ para indicar que pueden ser utilizados en nuevos intercambios de información entre candidatos. Aunque inicialmente la variable $set\_st\_avs_i$ es un conjunto vacío, durante la ejecución del algoritmo pueden incorporarse y/o eliminarse elementos en este conjunto.

La figura 4.2 resume la definición formal de las variables de estado del autómata A. En esta definición aparecen todas las variables que se han descrito anteriormente y los mensajes que pueden circular por cualquiera de los canales de comunicación del sistema.

Estado s del autómata del algoritmo solución A, $s \in states(A)$:

$\forall$ i $\in \mathcal{N}$:
  status.alg$_i$ $\in$ {active, blocked, candidate, dummy, victim, aborted}
  status.id$_i$ $\in$ {known, unknown}
  ta$_i$ $\in \mathbb{N}$
  t_unk$_i$ $\in \{-1\} \cup \mathbb{N}$
  sim_id$_i$ $\in$ T
  cand_succ$_i$ $\in$ T $\cup$ NULL
  setPred$_i$ $\subseteq 2^P$
  setPredToInf$_i$ $\subseteq 2^P$
  cont$_i$ $\in \mathbb{N}$
  nofirstAVS$_i$ $\in$ {true, false}
  inf_need$_i$ $\in$ {true, false}
  st_alg$_i$ $\in$ {(sid, path): sid $\in$ T $\wedge$ path $\in$ P$^+$ $\cup$ NULL}
  st_avsrsp$_i$ $\in$ {(t, sid, path): t $\in \mathbb{N}$ $\wedge$ sid $\in$ T $\wedge$ path $\in$ P$^+$ $\cup$ NULL}
  set_st_avs$_i$ $\in$ {(sid, t, path, bool): sid $\in$ T $\wedge$ t $\in \mathbb{N}$ $\wedge$ path $\in$ P$^+$ $\wedge$ bool $\in$ {true, false}}

$\forall$ {i,j} $\subseteq \mathcal{N}$:
  m $\in$ channel(i,j) | m $\in$ M

Figura 4.2: Definición del estado del autómata A.

Estado inicial de A:

El estado inicial de A refleja la situación de puesta en marcha del sistema. La definición formal del estado inicial $s_0 \in start(A)$ se proporciona en la figura 4.3. Puede observarse que en esta figura aparecen todas las variables que intervienen en la definición del estado del autómata del algoritmo. Los valores que adoptan inicialmente

Estado inicial (s_0) del autómata del algoritmo solución, A:

$\forall$ i $\in \mathcal{N}$:
  s_0.status.alg$_i$ = active
  s_0.status.id$_i$ = known
  s_0.ta$_i$ = 1
  s_0.t_unk$_i$ = -1
  s_0.sim_id$_i$ = (i, 1, ϵ)
  s_0.cand_succ$_i$ = NULL
  s_0.setPred$_i$ = $\emptyset$
  s_0.setPredToInf$_i$ = $\emptyset$
  s_0.cont$_i$ = 1
  s_0.nofirstAVS$_i$ = true
  s_0.inf_need$_i$ = false
  s_0.st_alg$_i$ = NULL
  s_0.st_avsrsp$_i$ = NULL
  s_0.set_st_avs$_i$ = $\emptyset$

$\forall$ {i,j} $\subseteq \mathcal{N}$:
  s_0.channel(i,j) = $\emptyset$
  s_0.channel(j,i) = $\emptyset$

Figura 4.3: Definición del estado inicial del autómata A.

estas variables permiten modelar el estado de reposo del autómata A en el que, básicamente, no existen relaciones de espera entre los nodos del sistema y no hay ningún mensaje en los canales de comunicación.

4.2.2. Signatura de las acciones del autómata A.

La signatura correspondiente al autómata del algoritmo solución, A, contiene además de las acciones de comunicación con el gestor, una serie de acciones internas. Al igual que en el autómata A_0 definido para la especificación del problema, las acciones de entrada tienen que ver con la aparición y desaparición de relaciones de espera en el sistema y la única acción de salida es la que informa de los abortos de los nodos en el sistema. Sin embargo, son las acciones internas las que modelan la parte fundamental del funcionamiento del algoritmo, esto es, las tareas propias de la detección de interbloqueos que se forman en el sistema.

En la figura 4.4 se puede observar la definición formal de la signatura de las acciones del autómata A, $sig(A)$.

$$
\begin{array}{l}
\texttt{Input}(A) = \{\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{t} \in \mathbb{N}: \\
\qquad \texttt{StartAddArc}_i\texttt{(j)},\ \texttt{EndAddArc}_i\texttt{(j,t)},\ \texttt{StartDelArc}_i\texttt{(j,t)},\ \texttt{EndDelArc}_i\texttt{(j)}\} \\
\\
\texttt{Internal}(A) = \{\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j}: \\
\qquad \texttt{initiate}_i,\ \texttt{rcvALG}_i\texttt{(j,m)} \mid \texttt{m} \in \texttt{M}_{ALG},\ \texttt{firstAVS}_i, \\
\qquad \texttt{rcvAVS}_i\texttt{(j,m)} \mid \texttt{m} \in \texttt{M}_{AVS},\ \texttt{sndAVS}_i, \\
\qquad \texttt{rcvAVSRSP}_i\texttt{(j,m)} \mid \texttt{m} \in \texttt{M}_{AVSRSP},\ \texttt{sndAVSRSP}_i, \\
\qquad \texttt{dltINF}_i\texttt{(j,m)},\ \texttt{rcvINF}_i\texttt{(j,m)} \mid \texttt{m} \in \texttt{M}_{INF}\} \\
\\
\texttt{Output}(A) = \{\forall\ \texttt{i} \in \mathcal{N}:\ \texttt{Abort}_i\}
\end{array}
$$

Figura 4.4: Signatura de las acciones del autómata A.

4.2.3. Las acciones el autómata *A*

En esta sección se presentan las acciones o pasos del autómata A, $steps(A)$. Del mismo modo que en el capítulo anterior, las acciones están descritas mediante el esquema de precondición-efecto. Para facilitar la explicación de cada una de ellas se han agrupado según la siguiente clasificación:

- Grupo I: las acciones de entrada que reflejan tanto la aparición como la desaparición de una relación de espera en el medio: $StartAddArc_i(j)$, $EndAddArc_i(j, t)$, $StartDelArc_i(j, t)$ y $EndDelArc_i(j)$.
- Grupo II: la acción de salida: $Abort_i$.
- Grupo III: las acciones internas que tienen que ver con la iniciación y la búsqueda de otro nodo iniciador: $initiate_i$, $rcvALG_i(j, m)$ y $firstAVS_i$.
- Grupo IV: las acciones internas que generan y procesan mensajes AVS y $AVSRSP$: $sndAVS_i$, $rcvAVS_i(j, m)$, $sndAVSRSP_i$ y $rcvAVSRSP_i(j, m)$.

- Grupo V: las acciones internas que procesan los mensajes *INF*: $dltINF_i(j,\ m)$ y $rcvINF_i(j,\ m)$.

Las figuras 4.5, 4.6 y 4.7 muestran el grupo I de acciones. Estas transiciones son las encargadas de modelar los efectos que tienen para el algoritmo la incorporación y el borrado de esperas en el sistema.

Añadir una relación de espera entre dos nodos del sistema supone una modificación en la configuración del medio. El algoritmo de detección y resolución de interbloqueos para poder desarrollar su tarea con éxito debe estar al corriente de esos cambios en el medio y adaptarse al nuevo escenario inmediatamente. Por ello, las acciones del medio $StartAddArc_i(j)$ y $EndAddArc_i(j,\ t)$ tienen su equivalente en el algoritmo y los cambios del medio pueden trasladarse a valores en las variables propias del algoritmo.

```
StartAddArc_i(j)
precondiciones:
efectos:
IF (status.id_i = known) THEN
  IF (setPredToInf_i = ∅) THEN
    status.alg_i:= blocked
  ELSE
    status.alg_i:= candidate;
    nofirstAVS_i:= true;
    FOR (id,t) ∈ (setPred_i \ setPredToInf_i):
     snd (ALG, t, sim_id_i, (i,ta_i)) to id;
     setPredToInf_i:= setPredToInf_i ∪ {(id,t)}
    ENDFOR
  ENDIF
ELSE status.alg_i:= blocked;
ENDIF.
```

```
EndAddArc_i(j,t)
precondiciones:
efectos:
setPred_i:= setPred_i ∪ {(j,t)};
IF (status.alg_i ∈ {dummy, candidate}) THEN
  setPredToInf_i:= setPredToInf_i ∪ {(j,t)};
  IF (status.alg_i = dummy) THEN
    snd (ALG, t, st_alg_i.sid, st_alg_i.path) to j
  ELSEIF (status.alg_i = candidate) THEN
    snd (ALG, t, sim_id_i, (i,ta_i)) to j;
  ENDIF
ENDIF.
```

Figura 4.5: Acciones de entrada del autómata A (grupo I): $StartAddArc_i(j)$ y $EndAddArc_i(j,\ t)$.

Acción $StartAddArc_i(j)$:

Según la acción análoga en el medio, los nodos que tienen habilitada esta acción son nodos activos, $state_i = active$. Según la propiedad B.3.3 del apéndice B, ese

estado se corresponde con la variable de estado del algoritmo, $status.alg_i = active$. Al ejecutar la acción del algoritmo se distinguen hasta tres casos diferentes de actuación de acuerdo con los valores de las variables $status.id_i$ y $setPredToInf_i$.

- Caso 1: $status.id_i = known$ y $setPredToInf_i = \emptyset$. El nodo i se asimila a los nodos del sistema en estado inicial porque no hay constancia de ninguna comunicación con otros nodos del sistema. El efecto de la acción en este supuesto consiste en un cambio de estado del nodo, $status.alg_i = blocked$. En el resto de variables asociadas al nodo i no se producen modificaciones.

- Caso 2: $status.id_i = known$ y $setPredToInf_i \neq \emptyset$. El nodo antes de volver a estar activo participó en un proceso de detección porque en la variable $setPredToInf_i$ quedaron registrados todos los nodos con los que estableció una comunicación propia de la fase de reconocimiento inicial.

  En este supuesto el nodo i se bloquea por el nodo j, pero su estado, en lugar de pasar a *blocked*, pasa a $status.alg_i = candidate$. En estas circunstancias, el nodo i se va a comportar del mismo modo que un nodo que espontáneamente ejecuta una instancia del algoritmo de detección de interbloqueo. Por lo tanto, para que el nodo i se asimile a un nodo iniciador del algoritmo que ha transmitido mensajes, es preciso que su variable $nofirstAVS_i$ tome el valor *true* y se envíen mensajes de tipo ALG a todos aquellos nodos que le preceden y con los que aún no ha establecido comunicación. Una vez que se generan esos mensajes ALG se puede afirmar que $setPred_i$ y $setPredToInf_i$ tienen el mismo contenido, tal y como hubiese sucedido si el nodo i hubiera iniciado la ejecución del algoritmo.

- Caso 3: $status.id_i = unknown$. Ese valor de $status.id$ indica que el nodo i está a la espera de recibir información sobre la identidad simulada que podría tener que adoptar, si se viera inmerso en un proceso de detección de interbloqueo, tras un cambio en el grafo de esperas. En cuanto el nodo i adquiere una nueva

identidad simulada o se valida la que ya posee, se podrá asumir con garantías la detección. Mientras tanto esta acción permitirá que el nodo i se bloquee por el nodo j, resultando $status.alg_i = blocked$. Como $status.id_i$ no se ve modificado, el nodo i seguirá a la espera de una nueva identidad simulada sin procesar información relativa a la búsqueda de un nodo detector entre los candidatos existentes.

Acción $EndAddArc_i(j,\ t)$:

La precondición de la acción $EndAddArc_i(j,\ t)$ en el medio indica que un nodo en cualquiera de los estados posibles, excepto en estado $state_i = aborted$, equivalente a $status.alg_i = aborted$, según la propiedad del B.3.5 del apéndice B, tendrá habilitada la acción si el nodo predecesor j se bloqueó por el nodo i. Esta precondición simula la fase inicial de formación de una espera entre el nodo j y el nodo i (la espera estaría representada parcialmente en un grafo).

El efecto fundamental de la acción consiste en incluir al nodo j junto con su tiempo asociado en el conjunto de predecesores, $setPred_i$. En el grafo de esperas que representa los bloqueos de los nodos del sistema, este efecto supone que el nodo j ya tiene conocimiento de que el nodo i inició una espera por él y, por lo tanto, la espera queda completa.

Además, el nodo i incluirá el elemento $(j,\ t)$ en $setPredToInf_i$, si el nodo i participa en una detección de interbloqueo siendo *candidate* o *dummy*. En caso de que el nodo i sea un candidato a convertirse en detector de un interbloqueo, éste enviará un mensaje ALG a su nuevo predecesor (el nodo j) con su propia información: sim_id_i y una ruta en la que aparece como único elemento su propio identificador y tiempo de activación. El hecho de que $status.alg_i = dummy$ significa que el nodo i redirigió información de un nodo candidato y, por tanto, al añadir el par $(j,\ t)$ a $setPred_i$, el nodo i debe transmitir esa misma información al nuevo nodo predecesor j. Los datos que se retransmiten están ubicados en st_alg_i y son tanto el campo $st\_alg_i.sid$ como $st\_alg_i.path$. El envío

de mensajes ALG, en estos dos casos, se completa como ya se ha comentado con la incorporación del nodo j en $setPredToInf_i$. Para el resto de posibles estados del nodo i estos efectos no se producen.

Las acciones de entrada del autómata que reflejan el borrado de una relación de espera del sistema son $StartDelArc_i(j,\ t)$ y $EndDelArc_i(j)$. La eliminación de la espera en el medio se realiza en dos fases ejecutando consecutivamente, y en ese orden, cada una de esas acciones. Las acciones del algoritmo, homónimas a las del gestor del sistema, interpretan los cambios producidos en el sistema y modifican las variables correspondientes del algoritmo, para que el funcionamiento de éste vaya acorde con la nueva configuración del sistema. La fase inicial del borrado de una relación de espera para el algoritmo sigue el esquema de efectos mostrado en la figura 4.6.

Acción $StartDelArc_i(j,\ t)$:

Esta acción tiene como efecto común a todos los posibles escenarios de ejecución la eliminación del par $(j,\ t)$ de $setPred_i$. Sólo cuando $setPred_i$ cuenta con más elementos que el par $(j,\ t)$ y el nodo i se encuentra en cualquiera de los estados siguientes: *dummy*, *candidate*, *victim* y *blocked-unknown*, el elemento $(j,\ t)$ se retirará también de $setPredToInf_i$. Todas estas situaciones se engloban en la condición de habilitación de la acción $StartDelArc_i(j,\ t)$ del medio que establece que $state_j = aborted$ (de acuerdo con la propiedad B.3.5 del apéndice B $state_j = aborted$ es equivalente a $status.alg_j = aborted$). Por lo tanto, este efecto se incluye para suprimir la parte final de la espera entre el nodo i y el nodo j que queda tras el aborto de este último.

Por otra parte, si $status.alg_i = active$, el nodo i podrá deshacerse de los mensajes almacenados en $set\_st\_avs_i$ que tengan como penúltimo elemento de la ruta al par $(j,\ t)$. Antes de eliminar esos mensajes se comprueba si la ruta resultante de suprimir el último elemento tiene algún otro par aparte de $(j,\ t)$. En caso afirmativo, se procederá a enviar mensajes AVS al nodo j, para reutilizar la información de las rutas

```
StartDelArc_i(j,t)
precondiciones:
efectos:
setPred_i:= setPred_i \ {(j,t)};
IF (status.alg_i ∈ {dummy, candidate}) ∨ (status.alg_i = blocked ∧ status.id_i = unknown) ∨
  (status.alg_i = victim ∧ setPred_i ≠ {(j,t)}) THEN
    setPredToInf_i:= setPredToInf_i \ {(j,t)};
ENDIF
IF (status.alg_i = active) THEN
  FOR m ∈ set_st_avs_i: penult(m.path) = (j,t):
    IF ((j,t) ≠ m.path - last(m.path)) THEN
      snd (AVS, m.sid, penult(m.path).ta, m.path - last(m.path), true) to penult(m.path).id;
      set_st_avs_i:= set_st_avs_i \ {m}
    ELSE set_st_avs_i:= set_st_avs_i \ {m}
    ENDIF
  ENDFOR
  IF (status.id_i = known) ∧ ({(j,t)} ∈ setPredToInf_i) THEN
    snd (INF, (sim_id_i.id, sim_id_i.ta, sim_id_i.nu·cont_i), t) to j;
    IF (setPred_i = ∅) THEN
      st_alg_i:= NULL;
      t_unk_i:= −1;
      inf_need_i:= false;
      nofirstAVS_i:= true;
      sim_id_i:= (i, ta_i, ϵ);
      cont_i:= 1
    ELSE cont_i:= cont_i + 1;
    ENDIF
    setPredToInf_i:= setPredToInf_i \ {(j,t)};
  ENDIF
ENDIF.
```

Figura 4.6: Acciones de entrada del autómata A (grupo I): $StartDelArc_i(j,\ t)$.

almacenadas en $set\_st\_avs_i$. El nuevo mensaje AVS dispondrá de la componente sid, del tiempo asociado al nodo j, de la ruta del correspondiente mensaje almacenado sin el último elemento y la variable fwd puesta a $true$. Sin embargo, cuando la ruta que se prepara para el envío del mensaje AVS sólo está formada por el elemento (j, t), se descarta generar el mensaje porque no tiene información útil y directamente se elimina el mensaje almacenado de $set\_st\_avs_i$.

Para un nodo i activo con $status.id_i = known$ y tal que (j, t) esté contenido en su $setPredToInf_i$, la acción $StartDelArc_i(j,\ t)$ proporciona otro efecto relacionado con la transmisión de su identidad simulada al nodo j. Como la espera establecida entre

ambos nodos comienza a romperse y el nodo i se comunicó en algún momento con el nodo j tras saber que le esperaba, se hará llegar al nodo j la identidad simulada del nodo i través de un mensaje *INF*. Esa identidad está constituida por los campos: $sim\_id_i.id$, $sim\_id_i.ta$ y $sim\_id_i.nu \cdot cont_i$. El tercer campo se construye de manera que sea diferente para todos y cada uno de los nodos a los que se les envía un mensaje similar. Para ello, se concatena a $sim\_id_i.nu$ el número natural, $cont_i$. Tras esta operación, se incrementa en una unidad el valor de $cont_i$, logrando así que la identidad simulada que viaja en el mensaje *INF* sea distinta siempre que un nodo se elimina de $setPred_i$ y de $setPredToInf_i$. Después de formar el mensaje *INF* será posible retirar el par $(j,\ t)$ de $setPredToInf_i$ sin perder información de utilidad sobre el nodo i para todos aquellos nodos que sabían de su bloqueo antes de que se activara.

Si el nodo que se retira de $setPred_i$ es el único, se alcanza una situación en la que no quedan más elementos en $setPred_i$. En consecuencia, $setPred_i = \emptyset$ y se inicializarán las variables del nodo i tales como: $st\_alg_i = NULL$, $t\_unk_i = -1$, $inf\_need_i = false$, $nofirstAVS_i = true$, $sim\_id_i = (i,\ ta_i,\ \epsilon)$ y $cont_i = 1$. El nodo i puede eliminar las informaciones almacenadas de instancias anteriores por lo que quedará en un estado similar al inicial hasta que se vea involucrado en una nueva instancia del algoritmo.

Para finalizar con el grupo de acciones de entrada del autómata A, se va a comentar la acción $EndDelArc_i(j)$. La activación total de un nodo dirigida por el gestor del sistema se traslada a las variables del algoritmo a través de los efectos de esta acción. En la figura 4.7 se pueden observar todos los cambios que afectan a las variables del proceso de detección de un interbloqueo cuando en el grafo de esperas un nodo se vuelve a activar.

Acción $EndDelArc_i(j)$:

Al ejecutarse esta acción el nodo i vuelve a su estado inicial, $status.alg_i = active$. Este efecto siempre viene acompañado de un cambio en su tiempo de activación que

```
EndDelArc_i(j)
precondiciones:
efectos:
IF (status.alg_i = dummy) THEN
  IF (setPredToInf_i = ∅) THEN
    st_alg_i:= NULL;
    t_unk_i:= −1
  ELSE status.id_i:= unknown
  ENDIF
ELSEIF (status.alg_i = candidate) THEN
  st_avsrsp_i:= NULL;
  cand_succ_i:= NULL;
  IF (inf_need_i = true) THEN
    status.id_i:= unknown
  ELSE
    st_alg_i:= NULL;
    t_unk_i:= −1;
  ENDIF
ENDIF
ta_i:= ta_i + 1;
status.alg_i:= active.
```

Figura 4.7: Acciones de entrada del autómata A (grupo I): $EndDelArc_i(j)$.

se tendrá como referencia temporal del nodo i en su próximo bloqueo. Con esto se consigue que ningún mensaje originado entre esperas anteriores provoque inconsistencias.

Por otra parte, el resto de efectos dependerá del estado que presenta el nodo i antes de la ejecución de la acción. Si $status.alg_i = dummy$ y el nodo i no ha podido transmitir un mensaje ALG por no tener predecesores, $setPredToInf_i = \emptyset$, entonces se procede a eliminar el contenido de st_alg_i y a inicializar la variable t_unk_i. Así se consigue que el nodo vuelva a estar preparado para participar en un nuevo intento de detección de interbloqueo. Estos mismos efectos también tienen lugar cuando $status.alg_i = candidate$ e $inf\_need_i = false$, esto es, el nodo i no ha generado un mensaje de tipo AVS o $AVSRSP$ en el que se hayan analizado las identidades simuladas de varios candidatos a detector de un interbloqueo.

Sin embargo, cuando el nodo i es *dummy* y su conjunto $setPredToInf_i$ posee al menos un elemento con el que se ha comunicado, el nodo i queda a la espera de descubrir cuál será su identidad simulada en la instancia activa del algoritmo de detección de interbloqueo. Esta situación de espera se representa con el valor $status.id_i = unknown$ que complementa al estado final del nodo i, $status.alg_i = active$. Así mismo, en caso de que el nodo i sea *candidate* y haya mandado un mensaje *AVS* o *AVSRSP* de acuerdo a las identidades simuladas de los nodos candidatos que conoce, $inf\_need_i = true$, la variable de estado relativa al conocimiento de la identidad simulada también se convierte en $status.id_i = unknown$. El valor *unknown* permite parar la transmisión de información del algoritmo hasta que el nodo i haya enviado a us predecesores la información que haga consistentes los mensajes que les envió previamente.

Por último, hay que comentar un efecto que se produce para los nodos candidatos que consiste en inicializar tanto la variable st_avsrsp_i como $cand\_succ_i$. La razón que conduce a eliminar sus valores es que la activación del nodo i hace falsa la información que contienen. Conforme las esperas entre los nodos sucesores del nodo i desaparecen secuencialmente, todo registro que conste de ellos será inservible para la detección de un interbloqueo posterior a su activación.

La única acción de salida del autómata es $Abort_i$. Los efectos de esta acción del algoritmo predisponen al medio para concluir la actividad de un nodo concreto del sistema. Ese nodo es el que en la ejecución del algoritmo se ha seleccionado como víctima (ver precondiciones de la acción en la figura 4.8). Desde el punto de vista del algoritmo, el nodo pasa a un estado que le incapacita para participar en nuevas detecciones de interbloqueo en el sistema. Por eso, los efectos de la acción consisten en la salvaguarda de toda la información que manejaba este nodo y la inicialización de las variables que tenía asociadas.

```
Abort_i
precondiciones: status.alg_i = victim.
efectos:
FOR m ∈ set_st_avs_i:
  IF (penult(m.path) ≠ m.path - last(m.path)) THEN
    snd (AVS, m.sid, penult(m.path).ta, m.path - last(m.path), true) to penult(m.path).id;
  ENDIF
ENDFOR
set_st_avs_i:= ∅
FOR (id,t) ∈ setPredToInf_i:
  snd (INF, (sim_id_i.id, sim_id_i.ta, sim_id_i.nu·cont_i), t) to id
  cont_i:= cont_i + 1;
ENDFOR
setPred_i:= ∅;
setPredToInf_i:= ∅;
st_avsrsp_i:= NULL;
nofirstAVS_i:= true;
ta_i:= ta_i + 1;
cont_i:= 1;
sim_id_i:= (i, ta_i, ϵ);
status.alg_i:= aborted.
```

Figura 4.8: Acción de salida del autómata A (grupo II): $Abort_i$.

Acción $Abort_i$:

Esta acción sólo la puede ejecutar un nodo que se haya convertido previamente en vítima. Eso implica que el nodo i detectó un interbloqueo, o en otras palabras, localizó un ciclo de esperas reales en el grafo que representa el funcionamiento del sistema. El nodo que detecta un interbloqueo en el sistema, según el principio de diseño de este algoritmo, dispondrá de la mayor identidad simulada de entre todos los nodos que conforman el ciclo o la situación de interbloqueo. La acción $Abort_i$ se encarga de resolver el interbloqueo del sistema eliminado el nodo i que lo ha detectado. Concluida la fase de resolución del interbloqueo, el nodo i no participará de forma activa en la evolución del sistema. Para poner de manifiesto esa situación, el estado del nodo i se transforma en $status.alg_i = aborted$. Aún siendo $status.alg_i = aborted$, el nodo i podría verse involucrado en la recepción de algún mensaje que surgiera con posterioridad a la resolución del interbloqueo, es decir, el nodo i participaría en

la evolución del sistema de forma pasiva. Si algún nodo presente en el interbloqueo resuelto invocara una nueva instancia del algoritmo, el nodo i podría estar presente en mensajes almacenados que se aprovecharían para una resolución más rápida y con menos mensajes de un nuevo interbloqueo.

Como el nodo i va a desaparecer del sistema de forma permanente tras la ejecución de esta acción, la mayor parte de sus efectos consistirán en restablecer todas las variables asociadas al nodo, que no se habían modificado al pasar a $status.alg_i = victim$, a sus valores iniciales correspondientes: $setPred_i = \emptyset$, $st\_avsrsp_i = NULL$, $nofirstAVS_i = true$, $cont_i = 1$. También se cambia su tiempo de activación y el valor de su identidad simulada (componente temporal actualizada) para que cualquier información que persista del nodo i en el resto de nodos del sistema pueda ser fácilmente rechazada al comprobar que la referencia temporal ha quedado obsoleta.

Antes de vaciar completamente el contenido de $set\_st\_avs_i$, se reciclan los mensajes que estaban almacenados. Para ello, se envían en forma de mensajes AVS con la componente $fwd = true$. Este valor indica que el mensaje AVS generado no se trata ni de un mensaje de reconocimiento entre dos candidatos (efecto de la acción $firstAVS_i$) ni de un mensaje localizador del candidato con mayor identidad simulada (efecto de la acción $sndAVS_i$). La nueva ruta del mensaje que se pone en el canal se obtiene a partir de la ruta del mensaje almacenado eliminando el último par correspondiente, precisamente, al nodo abortado. Como destino del nuevo mensaje AVS se elige el nodo que en la nueva ruta conformada queda en la última posición. Por último, cabe mencionar que si la ruta conformada sólo tuviera un único elemento, el efecto descrito quedaría anulado. El objeto de cancelar la generación de este tipo de mensajes AVS es evitar que circulen mensajes AVS en los canales de comunicación del sistema que no aportan información.

Otra variable asociada al nodo i que deja de tener sentido tras su aborto es la variable $setPredToInf_i$. Por eso, uno de los efectos de la acción es hacer que $setPredToInf_i$

$= \emptyset$. Los datos (identificador-tiempo) que están almacenados en dicha variable deben ser empleados antes de que sean borrados por efecto de $Abort_i$. El nodo i tiene que difundir su identidad simulada a todos los predecesores con los que estableció comunicación enviando mensajes *INF* con ese fin a esos nodos. Aunque la identidad simulada del nodo antes de ejecutar $Abort_i$ es única, transmite una identidad simulada diferente a cada uno de sus predecesores. Gracias a la variable $cont_i$ que se incrementa para cada uno de los nodos destino de los mensajes *INF*, se consigue que las identidades simuladas transmitidas tengan en común el primer y el segundo campo de la identidad simulada del nodo i, ($sim\_id_i.id$, $sim\_id_i.ta$), pero no el tercer campo de la identidad simulada del nodo i, $sim\_id_i.nu \cdot cont_i$. Una vez que los mensajes *INF* han sido mandados a los nodos de $setPredToInf_i$ para que puedan procesar convenientemente la información que el nodo i les envió a ellos o a través de ellos a otros nodos.

El grupo III de acciones se corresponde con las acciones internas del autómata que desarrollan la fase de iniciación ($initiate_i$) y las fases de reconocimiento y colaboración ($rcvALG_i(j, m)$ y $firstAVS_i$) del algoritmo. La figuras 4.9, 4.10 y 4.11 complementan las explicaciones dadas para cada una de estas acciones.

Acción $initiate_i$:

Sólo los nodos en estado *blocked-known* con predecesores pueden iniciar el algoritmo de manera espontánea (ver figura 4.9). En cuanto se ejecuta esta acción, el nodo i cambia de estado y pasa a ser candidato, $status.alg_i = candidate$, dejando patente que es un nodo que puede convertirse en el detector de un interbloqueo si lo hubiere en el sistema.

Otro efecto de esta acción consiste en que la variable booleana $nofirstAVS_i$ se inicialice al valor *true*. Esta asignación supone que el nodo i no se ha comparado con ningún otro candidato de identidad simulada inferior a la de él y, por lo tanto, no ha mandado ningún mensaje de tipo *AVS* para poner de manifiesto su superioridad.

```
initiate_i
precondiciones: status.alg_i = blocked ∧ status.id_i = known ∧ setPred_i ≠ ∅.
efectos:
status.alg_i:= candidate;
nofirstAVS_i:= true;
FOR (id,t) ∈ setPred_i:
  snd (ALG, t, sim_id_i, (i,ta_i)) to id;
ENDFOR
setPredToInf_i:= setPred_i.
```

Figura 4.9: Acciones internas del autómata A (grupo III): $initiate_i$.

Por otro lado, mediante el envío de mensajes ALG a todos los nodos que en ese instante están almacenados en el conjunto $setPred_i$, se difunde la información del nodo i: su identidad simulada, sim_id_i y una ruta en la que aparece como único elemento el par (identidad nodo i, tiempo de activación del nodo i). Como destino de cada mensaje ALG se elegirá el identificador de cada elemento de $setPred_i$ y el tiempo asociado a ese elemento se incluirá como parte del contenido del mensaje ALG. Ese tiempo asociado al nodo destino que es una componente del mensaje ALG será el que permita validar su vigencia en la recepción.

Finalmente, todo par (nodo, tiempo) del conjunto de predecesores del nodo i también será incluido en el conjunto $setPredToInf_i$, formado por los predecesores del nodo i a los que ha informado de su existencia y de sus posibilidades de llegar a ser el nodo detector de un interbloqueo. Este efecto implica que, tras la ejecución de la acción $initiate_i$, el conjunto $setPredToInf_i$ será una copia exacta del conjunto $setPred_i$.

Acción $rcvALG_i(j, m)$:

Esta acción queda habilitada cuando existe un mensaje de tipo ALG en el canal de comunicación, que tiene como origen el nodo j y como destino el nodo i. Además, el nodo i que recibe el mensaje ALG tiene pleno conocimiento de su identidad simulada en la ejecución del algoritmo.

```
rcvALG_i(j,m)
precondiciones: m ∈ channel(j,i) ∧ m.type = ALG ∧ status.id_i = known.
efectos:
channel(j,i):= channel(j,i) \ {m};
IF (m.ta = ta_i) THEN
  t_unk_i:= ta_i;
  IF (status.alg_i = blocked) THEN
    st_alg_i:= (m.sid, (i,ta_i)·m.path);
    status.alg_i:= dummy;
    FOR (id,t) ∈ setPred_i:
      snd (ALG, t, m.sid, (i,ta_i)·m.path) to id;
    ENDFOR
    setPredToInf_i:= setPred_i
  ELSEIF (status.alg_i = candidate) THEN
    IF (∃ t ∈ ℕ ∧ ∃ γ1 ∈ P+, γ2 ∈ P*: m.path = γ1·(i,t)· γ2 ∧ last(γ1) ∈ setPred_i) THEN
      status.alg_i:= victim;
      st_alg_i:= NULL;
      t_unk_i:= −1;
      cand_succ_i:= NULL;
      inf_need_i:= false
    ELSE
      st_alg_i:= (m.sid, m.path);
      cand_succ_i:= m.sid;
      IF (cand_succ_i > sim_id_i) THEN
        st_avsrsp_i:= (m.ta, m.sid, (i,ta_i)·m.path)
      ENDIF
    ENDIF
  ENDIF
ENDIF.
```

Figura 4.10: Acciones internas del autómata A (grupo III): $rcvALG_i(j, m)$.

Si se cumplen los requisitos anteriores, el mensaje ALG será retirado del canal. Antes de ser procesado, se comprobará si el tiempo del destino que se anotó en el mensaje ALG coincide con el tiempo de activación que en ese momento le corresponde al nodo i. En caso de que los tiempos comparados sean iguales, el mensaje ALG será tratado porque se considerará que toda la información que contiene es actual. Sin embargo, cuando los tiempos que hacen referencia al nodo i no coincidan, el mensaje será rechazado porque contará con información desfasada.

Una vez realizada la validación temporal del mensaje ALG recibido, se asignará el tiempo de activación del nodo a la variable t_unk_i para indicar el instante en el que

el nodo i recibió información del nodo al que espera de manera directa en el sistema. Con posterioridad, el mensaje ALG se tratará según sea el estado del nodo receptor *blocked* o *candidate* (véase la figura 4.10).

- Si el nodo i está bloqueado, $status.alg_i = blocked$, se almacenará en st_alg_i el campo del mensaje ALG que hace referencia a la identidad simulada del nodo que lo generó y además se copiará la ruta del mensaje $m.path$ antecedida por el identificador y el tiempo del nodo i. Al incorporarse el nodo i a esta ruta, se asume que la espera entre el nodo i y el primer nodo que figura en esa ruta es real.

  Además, en esta situación hay que asegurar que el nodo i ya no tenga ninguna oportunidad de iniciar una copia de algoritmo. Al recibir un mensaje ALG de su sucesor en el grafo de esperas, el nodo i debe quedar incapacitado para tener habilitada la acción $initiate_i$. Para ello, se cambia el estado del nodo y se convierte en *dummy*.

  Toda la información que se ha almacenado en st_alg_i se empleará para construir un mensaje ALG para cada uno de los nodos que el nodo i contenga en su variable $setPred_i$. Una vez entregados los mensajes ALG a los canales indicados por los identificadores de los nodos predecesores del nodo i, el conjunto $setPredToInf_i$ se igualará al conjunto $setPred_i$.

- Cuando el nodo i es candidato a detector de un interbloqueo en el sistema, $status.alg_i = candidate$, se comprueba primeramente si en la ruta del mensaje ALG recibido aparece ya el nodo i. Su aparición implica que en el sistema existe un interbloqueo. Sólo se confirma que hay un interbloqueo en el sistema cuando todas las esperas previas hasta alcanzar al nodo i son reales, o de manera equivalente a esta condición, cuando el par (nodo-tiempo) anterior al nodo i en la ruta del mensaje ALG pertenece al conjunto de predecesores del nodo i,

$setPred_i$. Nótese que si aparece (i, t) con un tiempo asociado diferente al actual, eso implica que todas las esperas a partir de él que están presentes en la ruta (en γ_2) han dejado de existir. Sin embargo, si $last(\gamma_2) \in setPred_i$ significa que el nodo i nunca se desbloqueó por lo que todas las esperas reflejadas en la primera parte de la ruta, γ_1 siguen existiendo.

Si hay presente un ciclo en la ruta del mensaje ALG, se designará al nodo i como nodo detector del interbloqueo y víctima para la resolución del interbloqueo. Algunas de las variables asociadas al nodo i tales como: st_alg_i, t_unk_i, $cand\_succ_i$ e inf_need_i, se inicializarán porque el nodo i a partir de ese instante no les dará ningún uso. En cambio, ante la inexistencia de un ciclo en la ruta del mensaje ALG, los efectos de la acción consitirán en almacenar el mensaje tal cual llega (campo *sid* y ruta) y cargar el término *sid* en la variable $cand\_succ_i$. Por último, el mensaje ALG completo se guardará en st_avsrps_i cuando la identidad simulada del nodo i sea inferior al candidato sucesor del mismo. En esta comparación se establece que el nodo candidato que ha recibido el mensaje ALG (nodo i) no va a competir más en la carrera por ser el detector del interbloqueo porque el nodo candidato que le sucede en el grafo esperas, $cand\_succ_i$, presenta una identidad simulada superior a la de él. A diferencia del mensaje que se copió en st_alg_i, el que se almacena en st_avsrsp_i incluye como primer elemento de la ruta al nodo i y su tiempo de activación. En cuanto el nodo i tenga conocimiento de un nodo candidato, que le preceda y que sea mayor que él, podrá decidir cuál de los candidatos, predecesor o sucesor, tiene posibilidades de convertirse en el detector de un interbloqueo que los incluya a todos ellos.

Acción $firstAVS_i$:

Únicamente podrá ejecutar la acción un nodo i que cumpla las siguientes precondiciones:

- $status.id_i = known$, el nodo i conoce su identidad simulada.
- $st\_alg_i \neq NULL$, el nodo i recibió y tiene almacenado un mensaje ALG.
- $nofirstAVS_i = true$, el nodo i no ha mandado todavía un mensaje AVS a un nodo candidato sucesor indicándole que cuenta con una identidad simulada superior a la que el candidato sucesor posee.
- $cand\_succ_i \neq NULL$, el nodo i conoce a un candidato sucesor.
- $cand\_succ_i < sim\_id_i$, la identidad simulada del nodo i es superior a la identidad del candidato sucesor en el grafo.

El efecto principal de esta acción es el envío de un mensaje AVS al nodo candidato que le sucede en el grafo de esperas y del que el nodo i sabe que es inferior a él. Como contenido del mensaje AVS se incluye la identidad simulada del nodo i, el tiempo del nodo al que va dirigido el mensaje, la ruta que almacena en su correspondiente st_alg_i precedida por su propio identificador y tiempo de activación, $(i,\ ta_i)$, y una marca $fwd = false$ que indica que el mensaje AVS no surge por la desactivación de candidatos sucesores y, por lo tanto, no está volviendo a recorrer los nodos de la ruta que contiene y ya ha analizado. El destino del mensaje coincide con el último nodo que aparece en la ruta de su st_alg_i. De este nodo se sabe que generó el mensaje ALG, el cual llegó hasta el nodo i siendo candidato. Cabe destacar que en el momento de la ejecución de la acción o cuando el mensaje AVS sea recibido y retirado del canal, podría suceder que el nodo ya no fuera candidato o fuera candidato con características diferentes a las que tenía en el instante en que se dio a conocer mediante el ALG que posteriormente recibió el nodo i.

En el momento en que se va a crear este mensaje AVS el nodo i deja constancia de ese hecho asignando a la variable booleana $nofirstAVS_i$ el valor *false*. El nodo i sólo podrá volver a tener en esta variable el valor *true* si en la evolución del sistema

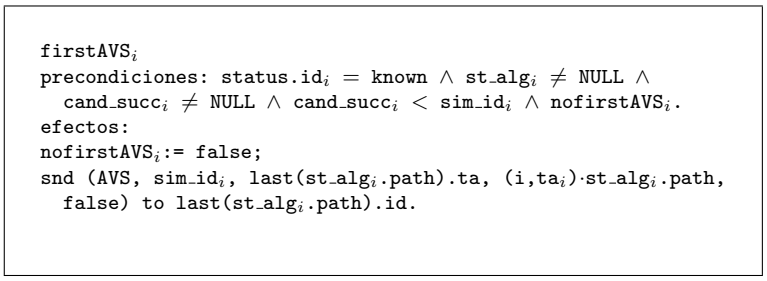

Figura 4.11: Acciones internas del autómata A (grupo III): *firstAVS*$_i$.

cambia su candidato sucesor en el grafo. Esto implicará que el nodo i se ha activado y se ha bloqueado de nuevo por un nuevo nodo o el mismo nodo pero con otra referencia temporal. Éste y el resto de efectos de esta acción se pueden observar en la figura 4.11.

El grupo IV de acciones internas que se va a explicar seguidamente tienen que ver con la recepción y envío de mensajes *AVS* y *AVSRSP* cuando varios nodos candidatos negocian por convertirse en el nodo que detecte el interbloqueo. En primer lugar, se presentan las acciones relacionadas con los mensajes *AVS* (ver figura 4.12) y posteriormente se tratarán las acciones relacionadas con el otro tipo de mensajes.

Acción *rcvAVS*$_i$(j, m):

Esta acción puede ejecutarse en dos supuestos distintos. En primer lugar, el nodo i recibirá los mensajes *AVS* cuyo campo *fwd* tenga el valor *false*. Que *m.fwd* sea igual a *false* implica que el mensaje *AVS* que se acepta del canal es un mensaje en el que la ruta que se transporta ha ido creciendo conforme se ha explorado el grafo de esperas y se ha combinado con otras rutas. Por otra parte, el nodo i también puede admitir mensajes *AVS* en los que el tiempo de activación del nodo i es superior al que tiene asociado en la ruta del mensaje.

```
rcvAVS_i(j,m)
precondiciones: m ∈ channel(j,i) ∧ m.type = AVS ∧ status.id_i = known ∧
  (m.fwd = false ∨ (ta_i > last(m.path).ta)).
efectos:
channel(j,i):= channel(j,i) \ {m};
IF (penult(m.path) ∈ setPred_i) THEN
  IF (sim_id_i > m.sid) THEN
    snd (AVSRSP, first(m.path).ta, sim_id_i, m.path) to first(m.path).id
  ELSEIF (sim_id_i ≤ m.sid) THEN
    IF (status.alg_i = candidate) ∧
    (first(m.path).id = last(m.path).id) = i) ∧ (sim_id_i = m.sid) THEN
      status.alg_i:= victim;
      st_alg_i:= NULL;
      t_unk_i:= −1;
      cand_succ_i:= NULL;
      inf_need_i:= false
    ELSE //no se cumple la situación de víctima//
      IF (∄ m/ ∈ set_st_avs_i: m/.path = γ1·m.path ∧ γ1 ≠ ϵ ∧ m.fwd = true) THEN
        set_st_avs_i:= set_st_avs_i ∪ {(m.sid, m.ta, m.path, false)}
        IF (∃ m/ ∈ set_st_avs_i: m.path = γ1·m/.path ∧ γ1 ≠ ϵ ∧ m.fwd = true) THEN
          set_st_avs_i:= set_st_avs_i \ {m/};
        ENDIF
      ENDIF
    ENDIF
  ENDIF
ELSEIF (penult(m.path) ∉ setPred_i)THEN
  IF (penult(m.path) ≠ m.path - last(m.path)) THEN
    snd (AVS, m.sid, penult(m.path).ta, m.path - last(m.path), true) to penult(m.path).id
  ENDIF
ENDIF.

sndAVS_i
precondiciones: status.id_i = known ∧ st_avsrsp_i ≠ NULL ∧ cand_succ_i ≤ sid ∧
  ∃ p ∈ P+, t ∈ ℕ, sid ∈ T: (sid, t, p, false) ∈ set_st_avs_i.
efectos:
set_st_avs_i:= set_st_avs_i \ {(sid, t, p, false)};
snd (AVS, sid, last(st_avsrsp_i.path).ta, (p - last(p))·st_avsrsp_i.path,
  false) to last(st_avsrsp_i.path).id;
inf_need_i:= true.
```

Figura 4.12: Acciones internas del autómata A (grupo IV): $rcvAVS_i(j, m)$ y $sndAVS_i$.

Es muy importante destacar que si el nodo i no tiene pleno conocimiento de la identidad simulada que representa en el proceso de detección ($status.id_i = known$) no podrá retirar del canal ningún mensaje AVS que vaya dirigido a él. Los mensajes INF tienen prioridad frente a los mensajes AVS.

Para procesar convenientemente el mensaje AVS se procederá a comprobar que el penúltimo elemento de la ruta y el nodo i mantienen una espera real. Esta prueba consiste simplemente en verificar que el penúltimo elemento de la ruta del mensaje AVS pertenece a $setPred_i$. En el supuesto de que no se cumpliera esta relación de pertenencia, el mensaje AVS se corregiría eliminado el último elemento de la ruta y se encaminaría al penúltimo elemento de la ruta. Como la idea de la creación de este mensaje AVS consiste en reutilizar toda la información posible que sea válida según la configuración real del grafo de esperas del sistema, a la componente booleana *fwd* del mensaje se le asignará el valor *true*. Al corregir la ruta de este tipo de mensajes AVS si resulta un mensaje AVS con sólo un par nodo-tiempo, el mensaje no llega a lanzarse al canal porque no contiene el mínimo de información requerida.

Si $penult(m.path) \in setPred_i$ es preciso asegurar que el mensaje AVS no cuenta con información que haya pasado a ser incorrecta por la evolución del sistema. En el caso de que la identidad simulada del nodo i sea mayor que la identidad simulada del candidato mayor extraído de la ruta del mensaje AVS, el mensaje AVS deberá transformarse en un mensaje $AVSRSP$ en el que la identidad simulada del nodo i se apunte como la mayor de los candidatos incluidos en la ruta. El destino del mensaje $AVSRSP$ reconvertido a partir de un mensaje AVS será el primer nodo de la ruta.

Después de estas comprobaciones se analiza si el mensaje debe convertir en víctima a su receptor. Esto sucede cuando el identificador del primero y del último nodo de la ruta coinciden con el nodo i que es el receptor del mensaje AVS. Al ser proclamado el nodo i como detector del interbloqueo presente en el sistema, será designado como la víctima propicia para que mediante su aborto, el interbloqueo quede resuelto. Una

vez que se selecciona al nodo i para solucionar el interbloqueo, $status_i = victim$ y previendo que tras su aborto el nodo i no actuará activamente en la evolución del sistema, se modifican las siguientes variables asociadas al nodo i: $st\_alg_i = NULL$, $t\_unk_i = -1$, $cand\_succ_i = NULL$ e $inf\_need_i = false$.

Cuando la recepción del mensaje AVS no conlleva el designio de una víctima, el mensaje AVS queda almacenado en $set\_st\_avs_i$ con su componente $bool$ puesta a $false$. Es posible que la ruta del mensaje AVS recibido con $m.fwd = true$ esté contenido en otro mensaje de los almacenados en $set\_st\_avs_i$. En ese supuesto, el mensaje AVS que recibe el nodo i no será incluido en $set\_st\_avs_i$. Por otro lado, si hubiera ya un mensaje AVS almacenado cuya ruta estuviera contenida en la ruta del menaje AVS que recibe el nodo i, se procederá a eliminar ese mensaje que estaba almacenado antes de la recepción del nuevo mensaje AVS.

Acción $sndAVS_i$:

Para que la ejecución de esta acción sea factible es necesario que:

- el nodo i conozca su identidad simulada en el proceso activo de detección de interbloqueo ($statud.id_i = known$)
- el nodo i tenga almacenada una ruta en st_avsrsp_i formada a partir de la exploración del grafo de esperas correspondiente a otros nodos candidatos que le suceden en la ruta.
- el nodo i tenga almacenada una ruta en $set\_st\_avs_i$ formada a partir de la exploración del grafo de esperas que él inició y que se corresponde con otros nodos candidatos que le preceden en la ruta. Se asume que el nodo i es el nodo candidato con mayor identidad simulada de entre todos los candidatos que están registrados en la ruta contenida en $set\_st\_avs_i$.
- la identidad del candidato sucesor del nodo i sea inferior o igual a la identidad

disponible en el mensaje de $set\_st\_avs_i$, que corresponde a un nodo que precede al nodo i.

Si se cumplen todos estos requisitos, el nodo i está en condiciones de confirmar al nodo sucesor que aparece en st_avsrsp_i que existe otro candidato con mayor identidad simulada que él según toda la información de la que dispone en este punto de la ejecución del algoritmo. Para llevar a cabo esta notificación, el nodo i genera un mensaje *AVS* compuesto de los siguientes campos:

- término *sid* del mensaje $set\_st\_avs_i$ analizado que se corresponde con el valor de la identidad simulada mayor de los candidatos explorados.
- tiempo del nodo al que va destinado el mensaje *AVS*. Como destino del mensaje se elige al nodo que ocupa la última posición de la ruta que contiene st_avsrsp_i.
- ruta formada por la concatenación de la ruta del mensaje $set\_st\_avs_i$, a la que previamente se le ha eliminado el último elemento, y de la ruta de st_avsrsp_i
- término *fwd* de valor *false* para indicar que este mensaje *AVS* no es de los que sirve para salvaguardar información cuando han tenido lugar activaciones en el grafo de esperas analizado.

El nodo i podrá recordar que envió un mensaje *AVS* para la búsqueda del candidato con mayor identidad simulada gracias a que otro efecto de la acción asigna el valor booleano *true* a la variable inf_need_i.

Finalmente, el mensaje de $set\_st\_avs_i$ utilizado para la formación del mensaje *AVS* se elimina de $set\_st\_avs_i$ para impedir que se generen nuevos mensajes que incluyan información ya utilizada o que se emplee información obsoleta en evoluciones dinámicas del sistema.

Siguiendo con el grupo IV de las acciones internas del autómata A, se comentan a continuación las acciones que se hacen cargo del tratamiento y de la generación de mensajes *AVSRSP*, $rcvAVSRSP_i(j,\ m)$ y $sndAVSRSP_i$ respectivamente. En la figura 4.13 aparecen escritas ambas acciones siguiendo el esquema de precondición-efecto.

```
rcvAVSRSP_i(j,m)
precondiciones: m ∈ channel(j,i) ∧ m.type = AVSRSP ∧ status.id_i = known.
efectos:
channel(j,i):= channel(j,i) \ {m};
IF (m.ta = ta_i) THEN
  IF (status.alg_i = candidate) THEN
    cand_succ_i:= m.sid;
    st_avsrsp_i:= (m.ta, m.sid, m.path)
  ENDIF
ENDIF.

sndAVSRSP_i
precondiciones: status.id_i = known ∧ st_avsrsp_i ≠ NULL ∧ cand_succ_i > sid) ∧
  ∃ p ∈ P+, t ∈ ℕ, sid ∈ T: (sid, t, p, false) ∈ set_st_avs_i.
efectos:
set_st_avs_i:= set_st_avs_i \ {(sid, t, p, false)} ∪ {(sid, t, p, true)};
snd (AVSRSP, first(p).ta, st_avsrsp_i.sid, (p - last(p))·st_avsrsp_i.path) to first(p).id;
inf_need_i:= true.
```

Figura 4.13: Acciones internas del autómata A (grupo IV): $rcvAVSRSP_i(j,\ m)$ y $sndAVSRSP_i$.

Acción $rcvAVSRSP_i(j,\ m)$:

Al ejecutarse esta acción, se retira del canal que va del nodo j al i el mensaje, m, de tipo *AVSRSP*. La característica temporal del mensaje se chequea para comprobar si coincide con el tiempo de activación del nodo i, en cuyo caso se procederá a almacenar el mensaje tal cual llega en st_avsrsp_i y se registrará, como nuevo candidato sucesor del nodo i, al elemento de la componente *sid* del mensaje *AVSRSP*. En este término del mensaje *AVSRSP* se localizaba la información relativa al candidato de mayor identidad simulada de los que se encontraban en la ruta estudiada. Si la característica temporal del mensaje *AVSRSP* no coincide con ta_i, el mensaje no se procesará porque incluye información de una configuración de esperas que ya no existe.

Acción *sndAVSRSP*$_i$:

Las condiciones de habiltación de esta acción son análogas a las de la acción *sndAVS*$_i$ a excepción de que, en este caso, *cand_succ*$_i$ > *sid*. En la ejecución de esta acción se forma un mensaje *AVSRSP* concatenando una ruta perteneciente a *set_st_avs*$_i$ con la ruta de *st_avsrsp*$_i$ a través del nodo *i*. En la ruta empleada de *set_st_avs*$_i$ el nodo *i* ocupa la última posición de la ruta y en la de *st_avsrsp*$_i$ el nodo *i* es el primer elemento de la ruta. La disposición del nodo *i* en ambas rutas garantiza que la ruta que se forma gracias a este nexo de unión quede perfectamente concatenada.

A diferencia de la acción *sndAVS*$_i$, el mensaje almacenado en *set_st_avs*$_i$ que se ha empleado en la generación del nuevo mensaje *AVSRSP* no se elimina. En lugar de ello, se modificará su componente *bool* pasando a valer *true*. Este cambio evitará que el proceso de detección explore de nuevo las rutas de las que dispone información tanto en *st_avsrsp*$_i$ como en el mensaje empleado de *set_st_avs*$_i$.

Al igual que en la formación de un mensaje *AVS*, la variable *inf_need*$_i$ toma el valor *true*. Este booleano recuerda que, en caso de que el nodo *i* se active y forme parte de una nueva configuración del grafo de esperas, debe paralizar los mensajes del algoritmo hasta que obtenga información de otro candidato sucesor y decida si adopta su identidad simulada para posteriores fases de la detección de un interbloqueo.

El último grupo de acciones internas que va a ser descrito, grupo V, se corresponde con las dos acciones encargadas de procesar mensajes de tipo *INF*. La primera de ellas, *rcvINF*$_i(j,\ m)$, es la que va a permitir que un nodo, involucrado en un proceso de detección de interbloqueos, adquiera una nueva identidad simulada. La segunda acción, *dltINF*$_i(j,\ m)$, surge como mecanismo para eliminar estos mensajes del canal de comunicaciones cuando han perdido su vigencia. En la figura 4.14 se pueden observar ambas transiciones.

```
rcvINF_i(j,m)
precondiciones: m ∈ channel(j,i) ∧ m.type = INF ∧
  status.id_i = unknown ∧ m.ta = t_unk_i.
efectos:
channel(j,i):= channel(j,i) \ {m};
IF (inf_need_i = false) THEN//antes de activarse i era dummy//
  sim_id_i:= m.sid
ELSEIF (inf_need_i = true) THEN //antes de activarse i era candidate//
  IF (m.sid > sim_id_i) THEN
    sim_id_i:= m.sid;
    FOR n ∈ set_st_avs_i: sim_id_i > n.sid
      snd (AVSRSP, first(n.path).ta, sim_id_i, n.path) to first(n.path).id;
      set_st_avs_i:= set_st_avs_i \ {n}
    ENDFOR
  ENDIF
  inf_need_i:= false;
ENDIF
FOR (id,t) ∈ setPredToInf_i \ setPred_i
  snd (INF, (sim_id_i.id, sim_id_i.ta, sim_id_i.nu·cont_i), t) to id;
  cont_i:= cont_i + 1;
  setPredToInf_i:= setPredToInf_i \ {(id,t)};
ENDFOR
status.id_i:= known;
st_alg_i:= NULL;
t_unk_i:= −1;
IF (setPredToInf_i = ∅) THEN
  nofirstAVS_i:= true;
  sim_id_i:= (i, ta_i, ϵ);
  cont_i:= 1;
ELSEIF (setPredToInf_i ≠ ∅) THEN
  IF (status.alg_i = blocked) THEN
    status.alg_i:= candidate;
    nofirstAVS_i:= true;
    FOR(id,t) ∈ setPred_i \ setPredToInf_i
      snd (ALG, t, sim_id_i, (i,ta_i)) to id;
      setPredToInf_i:= setPredToInf_i ∪ {(id,t)};
    ENDFOR
  ENDIF
ENDIF.

dltINF_i(j,m)
precondiciones: m ∈ channel(j,i) ∧ m.type = INF ∧ m.ta ≠ ta_i ∧
  (t_unk_i = −1 ∨ t_unk_i ≠ m.ta).
efectos:
channel(j,i):= channel(j,i) \ {m}.
```

Figura 4.14: Acciones del autómata de la solución (grupo V): $rcvINF_i(j,\ m)$ y $dltINF_i(j,\ m)$.

Acción $rcvINF_i(j,\ m)$:

Al ejecutar esta acción, el nodo i mantiene o adopta la identidad simulada que le permitirá seguir participando en un proceso detección iniciado. En consecuencia, este efecto vendrá acompañado de un cambio en la variable $status.id_i$.

Una vez que el mensaje *INF* es retirado del canal por el nodo i, siendo $status.id_i = unknown$ y el tiempo del mensaje *INF* coincidente con el tiempo anotado en t_unk_i, el nodo i tiene que analizar si su identidad simulada es correcta para continuar con ella en la instancia del algoritmo o si, por le contrario, debe aceptar como propia la que está incluida en el mensaje *INF*. De acuerdo con el valor de inf_need_i, existen distintas posibilidades:

- $inf\_need_i = false$. Este valor indica que el estado del nodo i antes de activarse era *dummy*. Como el nodo i al activarse quedó con $status.id_i = unknown$, resulta obvio que se trataba de un nodo con predecesores a los que informar. Si este nodo volviera a bloquearse después de conocer su nueva identidad simulada o se hubiese bloqueado antes de conocerla, según los efectos de la acción $StartAddArc_i(j)$ o de la propia acción $rcvINF_i(j,\ m)$, el nodo i alcanzaría finalmente el estado *candidate*. En ese estado el valor de la identidad simulada es vital para la competición por llegar a ser el candidato detector de un interbloqueo. Por eso, el nodo i en este supuesto siempre adquirirá la identidad simulada que le proporcione el mensaje *INF*.

- $inf\_need_i = true$ y $m.sid > sim\_id_i$. En este caso el nodo i fue previamente *candidate* y adquiere la identidad simulada que viaja en el mensaje *INF*. Mientras el nodo i ha permanecido a la espera de recibir un mensaje *INF*, no ha podido almacenar mensajes nuevos en $set\_st\_avs_i$ porque la acción que produce ese efecto no está habilitada mientras $status.id_i = unknown$. A pesar de eso, el nodo i pudo recibir mensajes de tipo *AVS* antes de activarse y con el mensaje *INF*

dirigiéndose hacia él. Como los mensajes *AVS* que se almacenaron en dichas circunstancias podrían no estar cumpliendo los requisitos exigidos después de la adquisición del nuevo valor de identidad simulada, se debe comprobar que son válidos. Para ello, se compara la nueva identidad simulada del nodo i con el valor del campo *sid* del mensaje *AVS* almacenado en $set\_st\_avs_i$. Si la identidad simulada recién asignada a la variable sim_id_i es superior a la de los mensajes *AVS*, entonces el mensaje almacenado se reconvertirá a mensaje *AVSRSP* para salvar toda la información que contiene y posteriormente será eliminado. El mensaje *AVSRSP* se dirigirá al nodo que aparece en la primera posición de la ruta. Su ruta será la del mensaje almacenado que se retira y el valor de la componente *sid* será completado con la nueva sim_id_i.

- Cuando la identidad simulada que transporta el mensaje *INF*, $m_{INF}.sid$, no supera el valor de la identidad simulada del nodo i ($m.sid < sim\_id_i$), el nodo i mantiene su identidad simulada y tan sólo cambia el valor de inf_need_i a *false*.

Una vez que el nodo i tiene conocimiento de la identidad simulada que le corresponde, revisará si está obligado a propagar esa identidad simulada a otros nodos. El nodo i pudo liberar a alguno o a todos sus predecesores mientras su estado era $status.id_i = unknown$. En esa situación los nodos predecesores no se borraron del conjunto $setPredToInf_i$ porque precisan información para seguir vinculados al proceso de detección activo. Enviando mensajes *INF* a todos los elementos que están incluidos en $setPredToInf_i$ pero no en $setPred_i$, el nodo i logra transmitir su identidad simulada a todos ellos. Para una correcto funcionamiento del algoritmo, los nodos del sistema deben contar con una identidad simulada única. Por este motivo, a la hora de mandar la identidad simulada del nodo i a los nodos de $setPredToInf_i$ se añade un sufijo diferente al tercer campo de sim_id_i en cada envío. De esta manera, todos los predecesores a los que hay que informar reciben el mismo término de $sim\_id_i.id$

y de $sim\_id_i.ta$, pero un término $sim\_id_i.nu$ distinto debido a que se le concatena el valor numérico de $cont_i$ que se incrementa en una unidad cada vez que se elimina un elemento de $setPredToInf_i$.

Sea cual sea la identidad simulada que se obtiene en la ejecución de esta acción, tienen lugar una serie de efectos comunes a todas esas situaciones que son: $status.id_i = known$, $st\_alg_i = NULL$ y $t\_unk_i = -1$.

Después de los efectos anteriores es preciso comprobar si aún quedan elementos en el conjunto de predecesores del nodo i a los que hay que informar, $setPredToInf_i$. Si ya no hay ningún par nodo-tiempo a los que es necesario propagar la identidad simulada del nodo i ($setPredToInf_i = \emptyset$), se inicializarán las variables $nofirstAVS_i$, sim_id_i y $cont_i$. Sin embargo, cuando $setPredToInf_i \neq \emptyset$ y $status.alg_i = blocked$, el nodo i cambia su estado porque mientras permanecía a la espera de conocer su identidad de simulación se bloqueó de nuevo. En estas circunstancias el nodo i se incorpora al proceso de detección vigente como nodo candidato a detector de un interbloqueo. En consecuencia, se asumirá que la variable $nofirstAVS_i$ tiene el valor *true*. Por último, para adaptarse plenamente a la nueva condición de candidato, el nodo i enviará un mensaje ALG a todos y cada uno de los nodos que en el proceso de ajuste de su identidad simulada han pasado a formar parte de $setPred_i$ y no había tenido ocasión de contactar con ellos. Estos mensajes ALG son, por tanto, la tarjeta de presentación del nodo i. En su contenido será obligatorio que aparezca la identidad simulada con la que actuará a partir de este momento el nodo i y, además, la ruta del mensaje estará formada tan sólo por su identificador y tiempo de activación asociado. Tras el envío de estos mensajes ALG, tanto el conjunto $setPred_i$ como el conjunto $setPredToInf_i$ contendrán los mismos elementos.

Acción $dltINF_i(j, m)$:

Los mensajes INF surgen en el sistema tras el borrado de ciertas esperas en el grafo. Los nodos que se activan y que han participado activamente en el proceso

de detección de un interbloqueo quedan a la espera de este tipo de mensajes para conocer cuál será la identidad simulada que deban adoptar en la detección que se inició cuando estaban bloqueados. Si la evolución del sistema es tal que, en el momento de la recepción del mensaje *INF* que procede del nodo que fue sucesor directo, la información de la identidad simulada que transmite ya no es de utilidad entonces, se procederá a retirar el mensaje *INF* del canal. La acción $dltINF_i(j,\ m)$ mediante su único efecto acomete esta función. Como la información que viaja en el mensaje *INF* que se rechaza no tiene validez, el mensaje no se someterá a ningún procesado. Un mensaje *INF* deja de ser de interés para el desarrollo de un proceso de detección cuando el tiempo incluido en el mensaje no coincide con el tiempo del nodo al que iba destinado (nodo i) y además, o bien el nodo i se ha activado recuperando el valor inicial -1 de t_unk_i, o bien después de inicializarse la variable t_unk_i, el nodo i vuelve a bloquearse y a recibir un mensaje ALG que hace que la variable t_unk_i tome como valor un nuevo tiempo de activación.

4.2.4. Partición del autómata *A*

La relación de equivalencia o partición del autómata A, $part(A)$, se construye a partir del conjunto de acciones internas y de salida de A. Tal y como se muestra en la figura 4.15, las clases de equivalencia de la partición de A se disponen en ocho categorías distintas de acuerdo con las acciones que incluyen en su definición. La totalidad de estas clases persigue el objetivo común de impedir que la ejecución del algoritmo quede paralizada, pero cada uno de los conjuntos en los que se agrupan las clases de equivalencia tiene una finalidad concreta que se explica seguidamente.

La acción $Abort_i$ de cada nodo i representa una clase de equivalencia independiente. De esta manera se podrá afirmar que, una vez que el algoritmo declara a un nodo como el que debe abortar para resolver, siempre acaba abortando dicho nodo.

Resulta también evidente que la acción $initiate_i$ de cada nodo i tiene que constituir

$$
\begin{aligned}
\texttt{Part}(A) = \{&\forall\ \texttt{i} \in \mathcal{N}\texttt{: \{Abort}_i\texttt{\}},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{: \{initiate}_i\texttt{\}},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{: \{firstAVS}_i\texttt{\}},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{: \{sndAVS}_i\texttt{, sndAVSRSP}_i\texttt{\}},\\
&\forall\ \texttt{\{i,j\}} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{ALG}\texttt{: \{rcvALG}_i\texttt{(j,m)\}},\\
&\forall\ \texttt{\{i,j\}} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{AVSRSP}\texttt{: \{rcvAVSRSP}_i\texttt{(j,m)\}},\\
&\forall\ \texttt{\{i,j\}} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{AVS}\texttt{: \{rcvAVS}_i\texttt{(j,m)\}},\\
&\forall\ \texttt{\{i,j\}} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{INF}\texttt{: \{rcvINF}_i\texttt{(j,m), dltINF}_i\texttt{(j,m)\}\}}
\end{aligned}
$$

Figura 4.15: Partición del autómata A.

una clase de equivalencia independiente. Esto va a garantizar que, cuando un nodo queda interbloqueado, la copia del algoritmo en ese nodo iniciará su ejecución a no ser que reciba un mensaje ALG del nodo por el que está bloqueado. De igual modo, para permitir que un candidato, al que le llega información de un candidato de identidad inferior, señale con un mensaje AVS su superioridad y avance en la detección de un posible interbloqueo, la acción $firstAVS_i$ tiene que formar una nueva clase. Las acciones $sndAVS_i$ y $sndAVSRSP_i$ se han integrado en una misma clase porque todo nodo i que contenga información de otros candidatos (uno sucesor y otro predecesor) debe procesarla con el fin de difundirla y avanzar en el proceso de detección de un interbloqueo. En los tres tipos de clases que se acaban de mencionar, las acciones son ejecutadas tarde o temprano, a no ser que mientras tanto queden deshabilitadas para el nodo que corresponda.

Finalmente, la forma de asegurar que todos los mensajes llegarán a recibirse es estableciendo una clase de equivalencia para cada par de nodos y por cada una de las acciones encargadas de retirar mensajes de los canales de comunicación, esto es, $\{rcvALG_i(j, m)\}$, $\{rcvAVSRSP_i(j, m)\}$, $\{rcvAVS_i(j, m)\}$ y $\{rcvINF_i(j, m), dltINF_i(j, m)\}$. La última clase representada incluye dos acciones para conseguir que cualquier mensaje INF presente en un canal de comunicaciones no permanezca en él infinitamente. En este caso, según las características del nodo receptor, el mensaje INF se

retira y se procesa ($rcvINF_i(j,\ m)$) o bien se retira sin ningún efecto más ($dltINF_i(j,\ m)$).

4.3. El autómata del sistema, *S*

Al igual que ocurre en el autómata A_0, las acciones de A suponen que el medio se comporta de una forma determinada. Por ejemplo, se supone que nodos con $status.alg_i = victim$ no realizan acciones como la de añadir una espera. Sin embargo, las características descritas en A no aseguran que ese comportamiento sea cierto. Por ello, se vuelve a definir un sistema, como la composición del gestor del sistema G, que limita algunos comportamientos posibles, y el algoritmo A. El autómata de la composición se denomina S y se define como $S = \prod(G,\ A)$. Los autómatas que se componen, G y A, son fuertemente compatibles. La compatibilidad de estos autómatas viene dada por el cumplimiento de las siguientes propiedades:

- $Internal(G) = \emptyset \Rightarrow Internal(G) \cap acts(A) = \emptyset$
- $acts(G) = ext(A) \Rightarrow Internal(A) \cap acts(G) = \emptyset$
- $Output(G) \cap Output(A) = Output(G) \cap Input(G) = \emptyset$

Estado de S:

La teoría de composición de autómatas define el conjunto de variables de estado del autómata S como la unión de los conjuntos de variables de estado de los dos autómatas componentes. De esta forma, un estado $s \in states(S)$ viene determinado por los valores que adoptan las variables que se emplean en la defición del estado del autómata A (figura 4.2) y las que participan en la definición del autómata del medio G (figura 3.1). Dado que se pueden establecer equivalencias entre las variables de G y las de A, en los estados alcanzables de S se pueden obviar las variables duplicadas y

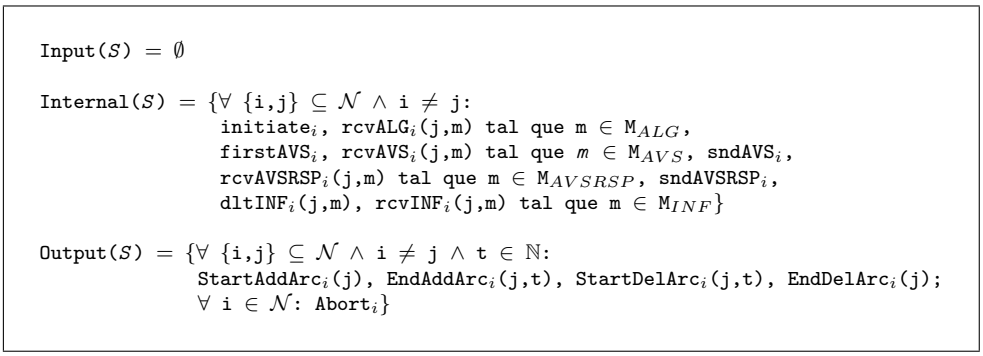

Figura 4.16: Signatura de las acciones del autómata S.

asumir que las variables que definen el estado de S son únicamente las que se emplean para definir el estado de A. Las relaciones entre algunas de las variables de los dos autómatas de la composición quedan demostradas en las propiedades del apéndice B.

De igual modo, el estado inicial de S, $start(S)$, queda definido por la unión de los valores iniciales de las variables que describen formalmente el estado inicial de G y de A, figuras 3.3 y 4.3 respectivamente. Además, se puede afirmar que $start(S)$ coincide con $start(A)$. Los valores iniciales que adoptan las variables de estado de S implican que cuando el sistema S parte del reposo: los recursos están libres, ningún proceso espera o retiene recursos y todavía no se ha enviado ningún mensaje. Por tanto, en el estado inicial $s_0 \in start(S)$, todos los nodos son activos, no hay esperas entre nodos, no hay mensajes en los canales de comunicaciones y las copias del algoritmo tampoco tienen información sobre otros nodos.

Signatura de S:

La signatura de acciones del autómata composición S está definida en la figura 4.16. Como puede observarse la signatura de S incluye la signatura de S_0. Esto evidencia que el autómata S trata de dar solución al problema que se especifica usando el autómata S_0. Los conjuntos de acciones de entrada y salida en S son los mismos que en S_0. Sin embargo, el conjunto de acciones internas de ambos autómatas difiere.

A diferencia de S_0, en el que no hay acciones internas, el autómata S incluye como acciones internas todas las acciones internas del autómata del algoritmo, A, que son realmente las encargadas de detectar posibles interbloqueos en el sistema.

Acciones de S:

El conjunto de pasos de S, $steps(S)$, se deduce a partir de los pasos de A y G.

```
StartAddArc_i(j)
precondiciones: state_i = active.
efectos:
state_i:= blocked;
blocker_i:= j;
set_waiters_j:= set_waiters_j ∪ {(i,t_activ_i,sent)};
IF (status.id_i = known) THEN
  IF (setPredToInf_i = ∅) THEN
    status.alg_i:= blocked
  ELSE
    status.alg_i:= candidate;
    nofirstAVS_i:= true;
    FOR (id,t) ∈ (setPred_i \ setPredToInf_i):
      snd (ALG, t, sim_id_i, (i,ta_i)) to id;
      setPredToInf_i:= setPredToInf_i ∪ {(id,t)}
    ENDFOR
  ENDIF
ELSE status.alg_i:= blocked
ENDIF.

  EndAddArc_i(j,t)
  precondiciones:state_i ≠ aborted ∧
    (j,t,sent) ∈ set_waiters_i.
  efectos:
  set_waiters_i:= set_waiters_i ∪ {(j,t,received)};
  set_waiters_i:= set_waiters_i \ {(j,t,sent)};
  setPred_i:= setPred_i ∪ {(j,t)};
  IF (status.id_i = known) ∧
  (status.alg_i ∈ {dummy, candidate}) THEN
    setPredToInf_i:= setPredToInf_i ∪ {(j,t)};
    IF (status.alg_i = dummy) THEN
      snd (ALG, t, st_alg_i.sid, st_alg_i.path) to j;
    ELSEIF (status.alg_i = candidate) THEN
      snd (ALG, t, sim_id_i, (i,ta_i)) to j;
    ENDIF
  ENDIF.
```

Figura 4.17: Acciones de salida del autómata S: $StartAddArc_i(j)$ y $EndAddArc_i(j, t)$.

Su definición formal mediante el esquema precondiciones-efectos se obtiene a partir de la definición dada para $steps(A)$, incorporando en las acciones de $Input(A)$ las precondiciones y los efectos que cada acción tiene en $Output(G)$. Del mismo modo, los efectos de la acción de $Input(G)$ también se añaden a los efectos de la acción correspondiente en $Output(A)$.

```
StartDelArc_i(j,t)
precondiciones:  (j,t,received) ∈ set_waiters_i ∧
  (state_i = active ∨ state_j = aborted).
efectos:
set_waiters_i:= set_waiters_i \ {(j,t,received)};
IF (state_i = active) THEN
  set_waiters_i:= set_waiters_i ∪ {(j,t,released)}
ENDIF;
setPred_i:= setPred_i \ {(j,t)};
IF (status.alg_i ∈ {dummy, candidate}) ∨ (status.alg_i = blocked ∧ status.id_i = unknown) ∨
(status.alg_i = victim ∧ setPred_i ≠ {(j,t)}) THEN
  setPredToInf_i:= setPredToInf_i \ {(j,t)};
ENDIF
IF (status.alg_i = active) THEN
  FOR m ∈ set_st_avs_i: penult(m.path) = {(j, t)}:
    IF (<j,t> ≠ m.path - last(m.path)) THEN
      snd (AVS, m.sid, penult(m.path).ta, m.path - last(m.path), true) to penult(m.path).id;
      set_st_avs_i:= set_st_avs_i \ {m}
    ELSEIF (<j,t> = m.path - last(m.path)) THEN
      set_st_avs_i:= set_st_avs_i \ {m}
    ENDIF
  ENDFOR
  IF (status.id_i = known) ∧ ({(j,t)} ∈ setPredToInf_i) THEN
    snd (INF, (sim_id_i.id, sim_id_i.ta, sim_id_i.nu·cont_i), t) to j;
    IF (setPred_i = ∅) THEN
      st_alg_i:= NULL;
      t_unk_i:= −1;
      inf_need_i:= false;
      nofirstAVS_i:= true;
      sim_id_i:= (i, ta_i, ϵ);
      cont_i:= 1
    ELSE cont_i:= cont_i + 1;
    ENDIF
    setPredToInf_i:= setPredToInf_i \ {(j,t)};
  ENDIF
ENDIF.
```

Figura 4.18: Acción de salida del autómata S: $StartDelArc_i(j,\ t)$.

```
EndDelArc_i(j)
precondiciones:  blocker_i = j ∧
  ((i,t_activ_i,released) ∈ set_waiters_j ∨ state_j = aborted).
efectos:
IF (state_j ≠ aborted) THEN
  set_waiters_j:= set_waiters_j \ {(i,t_activ_i,released)}
ELSEIF ((i,t_activ_i,sent) ∈ set_waiters_j) THEN
  set_waiters_j:= set_waiters_j \ {(i,t_activ_i,sent)}
ENDIF;
state_i:= active;
blocker_i:= NULL;
t_activ_i:=t_activ_i + 1;
IF (status.alg_i = dummy) THEN
  IF (setPredToInf_i = ∅) THEN
    st_alg_i:= NULL;
    t_unk_i:= −1
  ELSE status.id_i:= unknown
  ENDIF
ELSEIF (status.alg_i = candidate) THEN
  st_avsrsp_i:= NULL;
  cand_succ_i:= NULL;
  IF (inf_need_i = true) THEN
    status.id_i:= unknown
  ELSE
    st_alg_i:= NULL;
    t_unk_i:= −1;
  ENDIF
ENDIF
ta_i:= ta_i + 1;
status.alg_i:= active.
```

Figura 4.19: Acción de salida del autómata S: $EndDelArc_i(j)$.

Por lo tanto, como las acciones $StartAddArc_i(j)$, $EndAddArc_i(j, t)$, $StartDelArc_i(j, t)$, $EndDelArc_i(j)$ y $Abort_i$ son las únicas que cambian respecto a la definición de $steps(A)$, sólo se va a mostrar la definición de estas acciones (véanse las figuras 4.17, 4.18, 4.19 y 4.20). Para evidenciar la incorporación de las precondiciones y los efectos que proceden de las acciones del autómata G, se ha decidido escribir en cursiva estas aportaciones en la definición final de las correspondientes acciones de S.

Obviamente, si se consideran sólo estado alcanzables de S, las condiciones y efectos expresados con variables propias de las acciones de G se podrían reescribir con

```
Abort_i
precondiciones:  status.alg_i = victim.
efectos:
state_i := aborted;
blocker_i := NULL;
t_activ_i := t_activ_i + 1;
set_waiters_i := ∅;
FOR m ∈ set_st_avs_i:
  IF (penult(m.path) ≠ m.path - last(m.path)) THEN
    snd (AVS, m.sid, penult(m.path).ta, m.path - last(m.path), true) to penult(m.path).id;
  ENDIF
ENDFOR
set_st_avs_i := ∅
FOR (id,t) ∈ setPredToInf_i:
  snd (INF,(sim_id_i.id, sim_id_i.ta, sim_id_i.nu·cont_i), t) to id;
  cont_i := cont_i + 1;
ENDFOR
setPred_i := ∅;
setPredToInf_i := ∅;
st_avsrsp_i := NULL;
nofirstAVS_i := true;
ta_i := ta_i + 1;
cont_i := 1;
sim_id_i := (i, ta_i, ϵ);
status.alg_i := aborted.
```

Figura 4.20: Acción de salida del autómata S: $Abort_i$.

variables del algoritmo. Este cambio de notación en las acciones mostradas no afectaría al funcionamiento del autómata S y simplificaría su definición. Por ejemplo, la precondición de $StartAddArc_i(j)$ (ver figura 4.17), $state_i = active$, se sustituiría por $status.alg_i = active$. Asimismo, la precondición de $StartDelArc_i(j,\ t)$: $(j,\ t,\ received) \in set\_waiters_i \wedge (state_i = active \vee state_j = aborted)$ pasaría a ser $(j,\ t) \in setPred_i \wedge (status.alg_i = active \vee status.alg_j = aborted)$, tal y como se observa en la figura 4.18.

Respecto a los efectos se podría cambiar la variable t_activ_i por ta_i tanto en $Abort_i$ como en $EndDelArc_i(j)$, permitiendo en ambos casos eliminar efectos que aparecen duplicados en la definición de estas acciones en S. Puede comprobarse que en los estados alcanzables de S ambas versiones de las acciones coinciden. Las demostraciones de las propiedades que establecen estas equivalencias pueden localizarse en el apéndice B de esta tesis (ver las propiedades de B.3.1 a B.3.5). La equivalencia de otras

variables del medio como $set\_waiters_i$ y $blocker_i$ no se mencionan por la dificultad que entrañan.

Partición de S:

De acuerdo con la teoría de Autómatas de Entrada/Salida, la partición de un autómata resultante de la composición de autómatas fuertemente compatibles se define como la unión de las particiones de los autómatas componentes. Por consiguiente, la partición del autómata S, *Part(S)*, se obtiene de la unión de las particiones de sus autómatas componentes G y A, esto es, $Part(S) = Part(G) \cup Part(A)$. La unión de las particiones de los autómatas componentes implica que una ejecución equitativa del autómata composición S es equitativa con todas las clases de sus componentes G y A. La partición de G, autómata del gestor del sistema, puede observarse en la figura 3.11 del capítulo anterior y la partición de A se ha descrito en este mismo capítulo (ver figura 4.15).

Asumiendo que el sistema, S, evoluciona verificando una propiedad de equidad débil sobre su partición, se garantiza que, si existe una clase de $Part(S)$ que está continuamente habilitada, finalmente se ejecuta alguna de las acciones de la clase o deja de estar habilitada. Por tanto, al imponer la partición del sistema que se muestra en la figura 4.21, se asegura el progreso del algoritmo de detección y resolución de interbloqueos y se establece la base sobre la que construir la prueba de corrección del criterio de viveza del algoritmo.

De los diez grupos en los que se organizan las clases de equivalencia que conforman $Part(S)$, sólo dos pertenecen a la partición de G. Las acciones que pertenecen a estas dos clases tienen que ver con la adición y la eliminación de relaciones de espera. La acción $StartAddArc_i(j)$ constituye una clase de equivalencia por si sola y las acciones $EndAddArc_i(j)$, $StartDelArc_i(j,\ t)$ y $EndDelArc_i(j)$ se agrupan en otra clase de equivalencia.

$$
\begin{aligned}
\texttt{Part}(S) = \{&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j}\texttt{:}\{\texttt{StartAddArc}_i\texttt{(j)}\},\\
&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \texttt{t} \in \mathbb{N}\texttt{:}\ \{\texttt{EndAddArc}_i\texttt{(j,t)},\ \texttt{StartDelArc}_i\texttt{(j,t)},\ \texttt{EndDelArc}_i\texttt{(j)}\},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{:}\ \{\texttt{Abort}_i\},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{:}\ \{\texttt{initiate}_i\},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{:}\ \{\texttt{firstAVS}_i\},\\
&\forall\ \texttt{i} \in \mathcal{N}\texttt{:}\ \{\texttt{sndAVS}_i\texttt{,}\ \texttt{sndAVSRSP}_i\},\\
&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{ALG}\texttt{:}\ \{\texttt{rcvALG}_i\texttt{(j,m)}\},\\
&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{AVSRSP}\texttt{:}\ \{\texttt{rcvAVSRSP}_i\texttt{(j,m)}\},\\
&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{AVS}\texttt{:}\ \{\texttt{rcvAVS}_i\texttt{(j,m)}\},\\
&\forall\ \{\texttt{i,j}\} \subseteq \mathcal{N} \wedge \texttt{i} \neq \texttt{j} \wedge \forall\ \texttt{m} \in \texttt{M}_{INF}\texttt{:}\ \{\texttt{rcvINF}_i\texttt{(j,m)},\ \texttt{dltINF}_i\texttt{(j,m)}\}\}
\end{aligned}
$$

Figura 4.21: Partición del autómata S.

Al establecer que la acción $StartAddArc_i(j)$ para cualquier par de nodos del sistema se corresponda con una sola clase de equivalencia, se impone que la acción esté habilitada en todo momento. Con ello, se pretende simplemente que el sistema esté en funcionamiento y los nodos puedan iniciar una relación de espera por otro nodo en cualquier instante. Si se piensa en un sistema con procesos y recursos, la clase equivale a que un proceso solicite libremente un recurso para desarrollar sus tareas cuando lo requiera.

A diferencia de $StartAddArc_i(j)$, cuya ejecución se ha dispuesto de forma espontánea, el resto de acciones están supeditadas a que el sistema gestor realice las operaciones de asignación y liberación de recursos mediante el envío y recepción de mensajes. Así pues, una vez que un proceso ha solicitado un recurso, las acciones de la otra clase permiten que esa solicitud progrese y/o se cancele. La existencia de las clases que proceden de $Part(A)$ se argumentó en la descripción formal del autómata A.

Dado que el autómata S, descrito en esta sección, trata de dar solución al problema especificado formalmente por el autómata S_0, la partición de S debe contener la partición de S_0. Este hecho es apreciable al comparar las figuras 3.23 y 4.21.

Capítulo 5

Ejemplos de ejecución del algoritmo

En el presente capítulo se muestran una serie de ejemplos de ejecución del algoritmo de detección y resolución de interbloqueo que se ha diseñado. Las ejecuciones que se exponen a continuación pretenden, básicamente, clarificar el funcionamiento del algoritmo. Además, las explicaciones que se proporcionan pueden llegar a convertirse en una ayuda a la hora de comprender algunos aspectos del formalismo empleado tanto en la especificación del algoritmo como en su demostración. Por otra parte, los casos de ejecución que aquí se analizan, permiten entender los mecanismos que se han tenido que implementar para adaptar una solución del problema del interbloqueo estática a un escenario dinámico.

Para representar los mensajes que se producen en los procesos de detección de todas las ejecuciones analizadas se ha utilizado el código de flechas de colores y trazos que muestra la figura 5.1.

5.1. Escenarios estáticos

Las ejecuciones más simples del algoritmo son las que se desarrollan en escenarios estáticos. Se denominan así a aquellas computaciones en las que el grafo de esperas

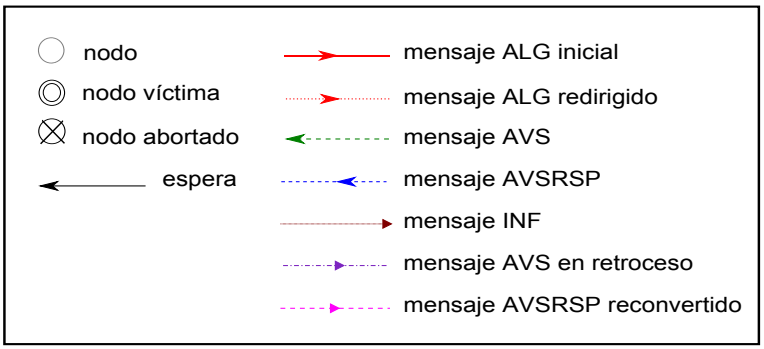

Figura 5.1: Leyenda de símbolos utilizados en los grafos de los ejemplos de ejecución.

que representa al sistema no sufre modificaciones mientras se ejecuta el algoritmo. Esto implica que la información que recopila el algoritmo durante su ejecución no se ve alterada. Si se piensa en las relaciones de espera que hay entre los nodos del interbloqueo a detectar, el carácter estático conlleva que no aparece ningún nuevo bloqueo o activación después de formarse el ciclo de esperas. Por esta razón, el tiempo de activación de un nodo que conforma el ciclo es una componente de la identidad del nodo que puede ser ignorada en la ejecución del algoritmo en un escenario estático.

En consecuencia, se puede afirmar que los tiempos de activación de los procesos (nodos del grafo) y los valores de las variables asociadas a la estructura del grafo no cambian durante la detección del ciclo. Cada una de las esperas del ciclo, se genera tras la ejecución de la acción $StartAddArc_i(j)$ seguida de la acción $EndAddArc_i(j,t)$. Tal y como se ha comentado, estas relaciones de espera quedan fijadas durante todo el proceso de detección del interbloqueo. Como la detección del ciclo, propiamente dicha, comienza cuando espontáneamente un nodo ejecuta la acción $initiate_i$, es posible que la fase de iniciación de la detección tenga lugar antes de que se haya terminado de formar el ciclo. De acuerdo con las precondiciones de la acción $initiate_i$, un nodo puede iniciar el algoritmo siempre que esté bloqueado, tenga conocimiento de su identidad (simulada) y, al menos, otro nodo esté bloqueado por él.

De todos los tipos de mensajes que aparecen en la figura 5.1, sólo aquéllos que

se representan con una punta de flecha no triangular se emplean para la detección y resolución de interbloqueos en los casos estáticos. Por eso, en los grafos de esta sección únicamente se podrán observar mensajes ALG iniciales y redirigidos (color rojo trazo continuo y discontinuo respectivamente), mensajes AVS (color verde) y mensajes $AVSRSP$ (color azul). Recuérdese en este punto que estos tres tipos de mensajes también se usan en el algoritmo de elección de líder [111] aunque con alguna funcionalidad diferente.

Cada vez que un nodo bloqueado redirige un mensaje ALG guarda la identidad del destino del mismo en $setPredToInf_i$. El almacenamiento de este dato, en caso de que el sistema evolucione dinámicamente y el camino seguido por el mensaje ALG del iniciador desaparezca, servirá para reconstruir las esperas desaparecidas y actualizar la información recogida en la propagación del mensaje. Consecuentemente, la variable $setPredToInf_i$ puede también ser ignorada en escenarios estáticos.

Para que un mensaje ALG sea retirado del canal y pueda ser procesado convenientemente, sólo se precisa que el estado del nodo que recibe el mensaje sea *known*, es decir, tenga conocimiento de su identidad simulada. En una detección realizada en un escenario estático, el estado de los nodos del ciclo siempre será *known* porque las relaciones de espera entre nodos no se ven modificadas. Por tanto, en una configuración estática durante el proceso de detección, ningún nodo tendrá que actualizar su identidad simulada de acuerdo a la evolución del sistema. En todos los casos de ejecución analizados para escenarios estáticos, la identidad simulada de un nodo se corresponde básicamente con la identidad del mismo. Por este motivo y, sin pérdida de generalidad, se puede hacer referencia indistintamente a la identidad simulada o a la identidad de un nodo. Esto implica que no habrá cambio alguno en la identidad simulada de los nodos del ciclo del ejemplo y cada uno conocerá su identidad simulada durante la ejecución del algoritmo.

5.1.1. Detección de un ciclo mediante un mensaje *ALG*

El primer ejemplo de detección, y el más sencillo de los ejemplos de este capítulo, supone la formación previa del ciclo que se observa en la figura 5.2.

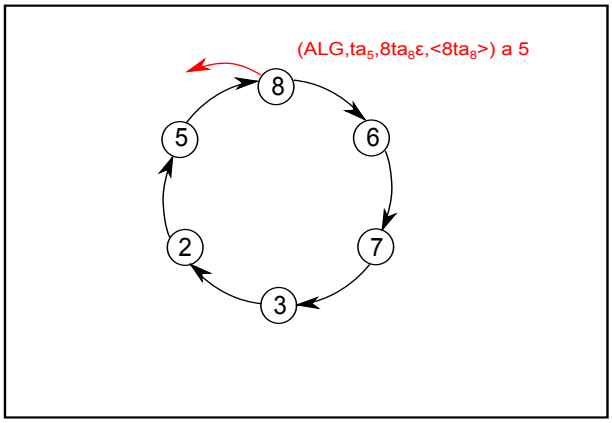

Figura 5.2: Detección con un mensaje ALG: fase de iniciación (escenario estático).

Obviamente, la descripción del proceso de detección de este primer ejemplo debe comenzarse explicando la fase de iniciación del algoritmo. Esta fase la pone en marcha el nodo del ciclo que lanza una instancia del algoritmo mediante la acción $initiate_i$. Se ha supuesto que este iniciador es el nodo 8 con identidad simulada $8ta_8\varepsilon$ (ver en el grafo de la figura 5.2). Los efectos de la acción $initiate_8$ consisten en cambiar el estado del nodo a candidato y en enviar un mensaje ALG a todos los nodos que lo preceden. Como en este caso, sólo hay un nodo predecesor (nodo 5), el mensaje ALG se dirigirá sólo al nodo 5.

La recepción del mensaje ALG marca el inicio de una nueva fase de la ejecución que trata de buscar otro nodo candidato que haya iniciado otra instancia del algoritmo. La figura 5.3 representa gráficamente esta fase de búsqueda de otros candidatos (iniciadores), que se lleva a cabo mediante la difusión de mensajes ALG entre los nodos predecesores. El envío y la recepción de un mensaje ALG es un proceso secuencial que se repite mientras el nodo que recibe el mensaje ALG sea un nodo bloqueado

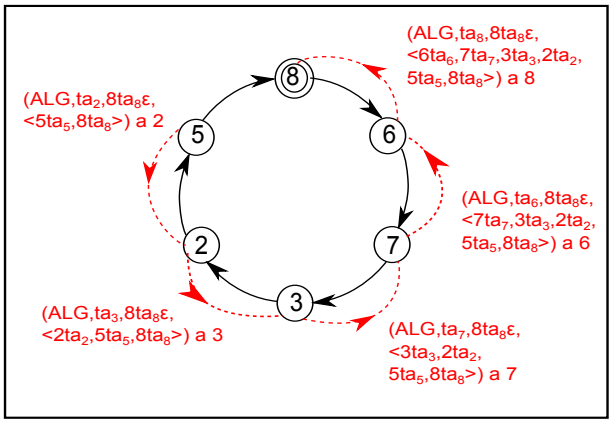

Figura 5.3: Detección con un mensaje ALG: fase de búsqueda de otro candidato y detección final (escenario estático).

que no haya ejecutado la acción $initiate_i$. Tal y como se observa en la figura 5.3, el nodo 5 manda un mensaje ALG a su predecesor, nodo 2. A su vez, el nodo 2 redirigirá ese mensaje ALG al nodo 3, después, el nodo 3 al nodo 7, el nodo 7 al nodo 6 y, finalmente, el nodo 6 a su predecesor, el nodo 8.

Los efectos de la acción $rcvALG_i(j,m)$ para un nodo bloqueado del ciclo se reducen a redirigir el mensaje ALG inicial entre sus predecesores, guardar en st_alg_i el mensaje recibido y cambiar su estado a *dummy*. El mensaje ALG que envía el nodo bloqueado se diferencia del que ha recibido o guardado en su ruta (campo *path* del mensaje). En la nueva ruta el nodo bloqueado incorpora al principio de la misma su identidad (identificador, tiempo de activación). Considerando el ciclo del ejemplo, el mensaje ALG que redirige el nodo 6 al nodo 8 contiene la siguiente ruta: $<6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8>$. Puede observarse en la figura 5.3 que la componente *sid* de todos los mensajes ALG que aparecen en este ejemplo incluye la identidad simulada del nodo iniciador del ciclo, $8ta_8\varepsilon$. Este dato, común a todos los mensajes ALG, identifica la instancia de la que forman parte.

La recepción del mensaje ALG por parte del nodo 8 constituye la fase de detección propiamente dicha. Al ser el nodo 8 un nodo candidato, la ruta del mensaje ALG que

le envía el nodo 6 es sometida a un análisis en profundidad. Como uno de los pares de la ruta del mensaje tiene como identificador al receptor del mensaje, nodo 8, y el par anterior a éste, $5ta_5$, está incluido en el conjunto de predecesores del nodo 8 ($setPred_8$), se puede concluir que el nodo 8 es el único iniciador del ciclo. Por ello, el nodo 8 puede autodesignarse víctima ya que ha sido capaz de detectar el interbloqueo presente en el sistema. El mensaje ALG que generó el nodo 8 al ejecutar el algoritmo ha vuelto a él, después de propagarse por los nodos bloqueados del ciclo de esperas. Por último, la fase de resolución del interbloqueo consiste en abortar la víctima señalada en la detección. Los efectos de la acción $Abort_8$ básicamente relegan al nodo 8 de su actividad en el sistema. La manera de apartar al nodo 8 del sistema consiste en romper las esperas que lo unen a su sucesor, el nodo 6, y a su predecesor, el nodo 5.

La tabla 5.1 muestra las acciones que participan en la detección mediante un mensaje ALG del escenario estático planteado en este primer ejemplo. Las acciones están ordenadas de acuerdo a la fase de ejecución de la que forman parte. Después de la tabla 5.1, se vuelven a mostrar los grafos del ejemplo pero esta vez de manera conjunta (figura 5.4). Con ello, se desea que el lector pueda asociar fácilmente las acciones del algoritmo a los mensajes que aparecen en la ejecución.

Fase	**Instancia 1-nodo 8**
iniciación	initiate_8
búsqueda de otro candidato	$\text{rcvALG}_5(8, m)$ $\text{rcvALG}_2(5, m)$ $\text{rcvALG}_3(2, m)$ $\text{rcvALG}_7(3, m)$ $\text{rcvALG}_6(7, m)$
detección	$\text{rcvALG}_8(6, m)$
resolución	Abort_8

Tabla 5.1: Secuencia de ejecución para la detección de un ciclo mediante mensaje ALG (escenario estático).

Si se dibuja esquemáticamente la ejecución del primer ejemplo estático propuesto, se aprecia que la detección de este interbloqueo consiste en el desarrollo de una única

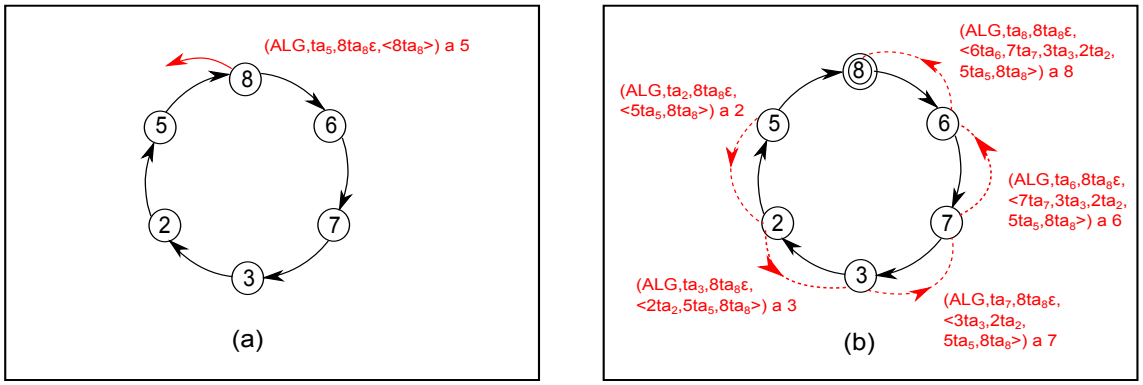

Figura 5.4: Detección con un mensaje ALG (escenario estático).

instancia del algoritmo tal y como indica la figura 5.5. Todo nodo bloqueado que recibe un mensaje ALG no puede iniciar una instancia del algoritmo a posteriori porque pasa a estado *dummy* como efecto de esa transición. Los nodos en estado *dummy*, mientras no se modifiquen las relaciones de espera del ciclo, no pueden convertirse en nodos candidatos. Por tanto, el nodo iniciador, nodo 8, es el único responsable de llevar a cabo la detección del ciclo existente. La instancia se inicia cuando el nodo 8 ejecuta la acción $initiate_8$ y finaliza con la resolución del interbloqueo, $Abort_8$.

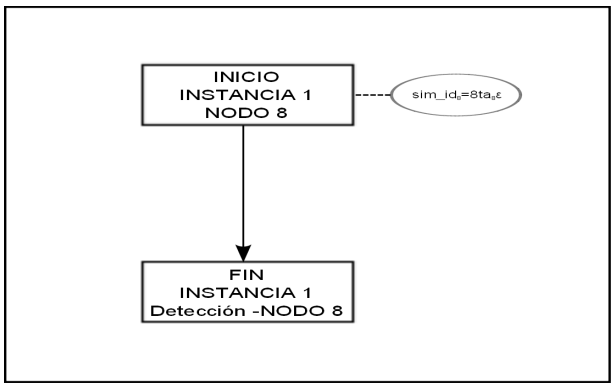

Figura 5.5: Diagrama de instancias del ejemplo de detección con mensaje ALG (caso estático).

5.1.2. Detección de un ciclo mediate un mensaje *AVS*

La detección y resolución de interbloqueo que se plantea en el siguiente ejemplo es la otra forma posible que el algoritmo diseñado tiene de abordar el problema del interbloqueo en escenarios estáticos. En el caso anterior sólo un nodo del ciclo lanzaba una instancia del algoritmo. Sin embargo, en este ejemplo, más de un nodo del ciclo, en concreto tres nodos, inician espontáneamente la ejecución del algoritmo. En la diferencia del número de iniciadores del algoritmo estriba que la detección se realice mediante un mensaje *ALG* o mediante un mensaje *AVS*.

Del mismo modo que en el caso previo, el proceso de detección del ciclo se inicia una vez que las relaciones de espera entre los nodos del mismo quedan establecidas. Además, durante la ejecución del algoritmo las esperas del ciclo no podrán cambiar. La secuencia de esperas que aparece en la figura 5.6.a es la configuración de nodos que se matendrá fija hasta que sea resuelto el interbloqueo.

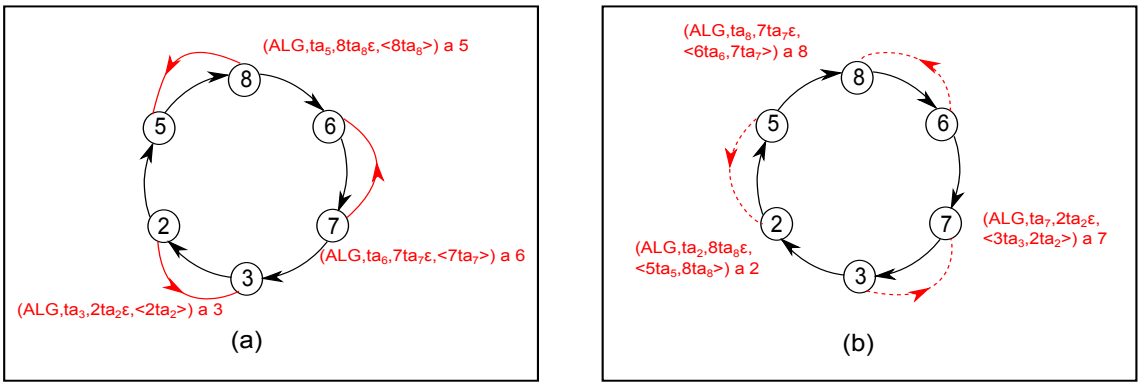

Figura 5.6: Detección con un mensaje *AVS*: fase de iniciación y búsqueda de otros candidatos (escenario estático).

La fase de iniciación de la detección del nuevo ejemplo comprende el lanzamiento de tres instancias del algoritmo, una por cada nodo iniciador. Se han elegido como iniciadores del ejemplo al nodo 8, al nodo 7 y al nodo 2 (ver en figura 5.6.a) . Estos nodos verifican la precondición de la acción $initiate_i$ y sus efectos pueden producirse

de forma simultánea o en distintos instantes del tiempo. Cada uno de estos nodos se convierte en candidato a detector del interbloqueo y envía un mensaje ALG para sondear a sus respectivos predecesores. Los nodos bloqueados que van encontrando a su paso estos mensajes ALG iniciales se transforman en *dummy* y se autoexcluyen del proceso de detección de un interbloqueo. A pesar de ello, la recepción de un mensaje ALG por parte de un nodo bloqueado supone la creación de un mensaje ALG con una ruta consistente con la propagación del mensaje ALG inicial hasta el nodo bloqueado receptor. Tal y como se explicó en la detección mediante mensaje ALG, los mensajes redirigidos que parten de un mismo nodo iniciador presentan todos el mismo valor de *m.sid*, esto es, la identidad simulada del nodo que originó el mensaje ALG en la acción $initiate_i$, aquí denominado ALG inicial.

A diferencia del escenario estático anterior, el mensaje ALG puede toparse con un nodo candidato distinto al nodo que lo generó inicialmente. En ese caso, los efectos de la acción $rcvALG_i(j,m)$ frenan el avance del mensaje ALG en la fase de búsqueda de otros candidatos del ciclo. El nodo 2 detiene el mensaje ALG que lanzó el nodo 8, el mensaje ALG correspondiente al nodo 2 es frenado por el nodo 7 y el del nodo 7 es parado por el nodo 8 (véase la figura 5.6.b). Los nodos candidatos receptores almacenan el mensaje ALG tal cual llega en la variable st_alg_i. Por ejemplo, st_alg_2 = (ta_2, $8ta_8\varepsilon$, $<5ta_5,8ta_8>$). Esto implica que el mensaje almacenado no contiene información del nodo candidato receptor correspondiente. A continuación, se produce otro efecto de la acción que tiene ver con la identificación del candidato sucesor más próximo. La variable $cand\_succ_i$ pasa a contener la componente *m.sid* del mensaje ALG. En el ejemplo que se está describiendo, los valores adoptados son: $cand\_succ_2$ = $8ta_8\varepsilon$, $cand\_succ_7 = 2ta_2\varepsilon$ y $cand\_succ_8 = 7ta_7\varepsilon$.

Posteriormente la identidad del candidato sucesor se compara con la identidad simulada del candidato receptor. A partir del resultado de esta comparación surgen diferentes opciones de ejecución. Si el candidato sucesor es mayor, el mensaje ALG

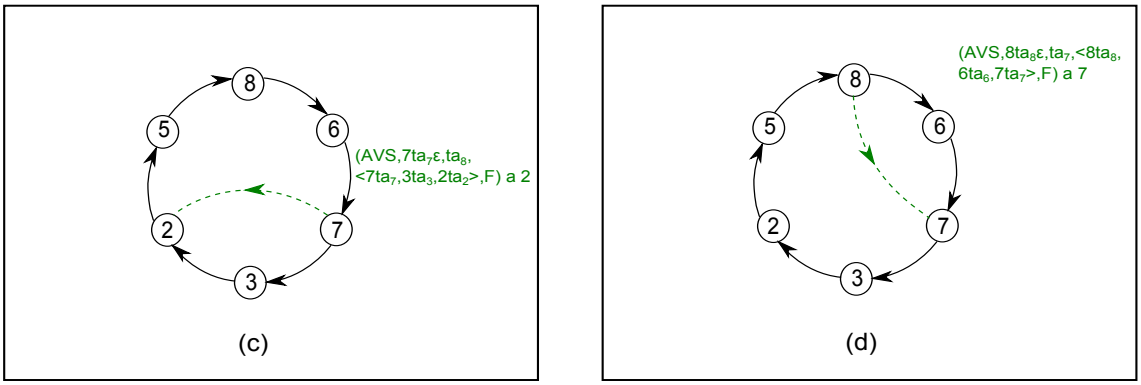

Figura 5.7: Detección con un mensaje *AVS*: fase de comparación directa de candidatos (escenario estático).

con el nodo receptor incorporado en la ruta se almacenará en una variable denominada st_avsrsp_i. Cuando el nodo receptor 2 realiza esta comparación, se completa su variable st_avsrsp_2 porque $cand\_succ_2 > sim\_id_2$ ($8ta_8\varepsilon > 2ta_2\varepsilon$). La información almacenada en st_avsrsp_2 contiene los datos (ta_2, $8ta_8\varepsilon$, $<2ta_2,5ta_5,8ta_8>$). El primer campo se corresponde con una referencia temporal que cobra sentido en escenarios dinámicos. El segundo dato coincide con la identidad del candidato sucesor que reconoce según la ruta que se registra en la tercera parte del mensaje.

En cambio, si la identidad simulada de nodo receptor es superior a la del candidato sucesor, queda habilitada la acción $firstAVS_i$. El objeto de esta acción es mandar un mensaje *AVS* que informará al candidato sucesor de su superioridad y, por lo tanto, le avisará de que no va llegar a ser detector del interbloqueo. El nodo 7 ejecuta $firstAVS_7$ con este fin y envía un mensaje *AVS* al nodo 2 (figura 5.7.c). En este mensaje *AVS* aparece la identidad simulada del nodo 7 y, como ruta, se antepone la identidad del nodo 7 al campo *path* de st_alg_7. Para señalar que entre los candidatos 7 y 2 ha habido una comunicación directa, a raíz de la recepción de un mensaje *ALG*, se debe poner a *true* la variable $nofirstAVS_i$ asociada al nodo ($nofirstAVS_7 = true$). Esta variable *booleana* controla si la fase de comparación directa de dos candidatos a detector,

se ha realizado durante la evolución de una configuración de esperas en el sistema. Otro efecto de la acción *firstAVS*$_i$ incluye un campo *m.fwd* con el valor *F* (*false*) al final del mensaje. Fijar el valor *F* hace posible distinguir el mensaje *AVS* que se ha generado de otros que surgen al considerar dinamismo en el sistema. Concretamente, la estructura del mensaje que se forma es (AVS, $7ta_7\varepsilon$, ta_2, $<7ta_7,3ta_3,2ta_2>$, F). Del mismo modo que el nodo 7, el nodo 8 también indica su superioridad al nodo 7 con un mensaje de tipo *AVS* (figura 5.7.d).

Los mensajes *AVS* se pueden extraer del canal siempre y cuando el nodo destino conozca su identidad simulada (*status.id*$_i$ = *known*). Además debe cumplirse que, bien sea un *AVS* procedente de la ejecución de la acción *firstAVS*$_i$ o de la acción *sndAVS*$_i$, bien el tiempo de activación del nodo receptor del mensaje *AVS* sea mayor que el tiempo que le corresponde a este nodo en la ruta del mensaje *AVS*. La primera condición se cumple obligatoriamente en un escenario estático y la característica temporal sólo se considera en caso de que hayan desaparecido esperas del sistema. En el ejemplo, la precondición de la acción *rcvAVS*$_i$(*j*,*m*) se verifica tanto para el nodo 2 como para el nodo 7. En ambos casos, los efectos de dicha acción retiran los mensajes *AVS* del canal y los almacenan en *set_st_avs*$_2$ y *set_st_avs*$_7$, respectivamente.

En este punto de la ejecución no quedan más candidatos del ciclo por descubrir y ya se han intercambiado todos los posibles mensajes *AVS*, resultado de la comparación directa de dos candidatos. Por ello, el proceso de detección prosigue con una nueva fase en la que algunos candidatos pueden comparar las identidades de sus candidatos vecinos y ordenarlas adecuadamente según sea mayor el candidato que le sucede o el que le precede. En el ejemplo actual, es el candidato 2 el que dispone de suficiente información para saber que debe abandonar la carrera a detector. Combinando la ruta que el nodo 2 tiene almacenada en *set_st_avs*$_2$, ($7ta_7\varepsilon$, ta_2, $<7ta_7,3ta_3,2ta_2>$, F), y en *st_avsrsp*$_2$, (ta_2, $8ta_8\varepsilon$, $<2ta_2,5ta_5,8ta_8>$), se construye un camino que, siguiendo las relaciones de espera existentes, conduce desde el candidato predecesor del nodo

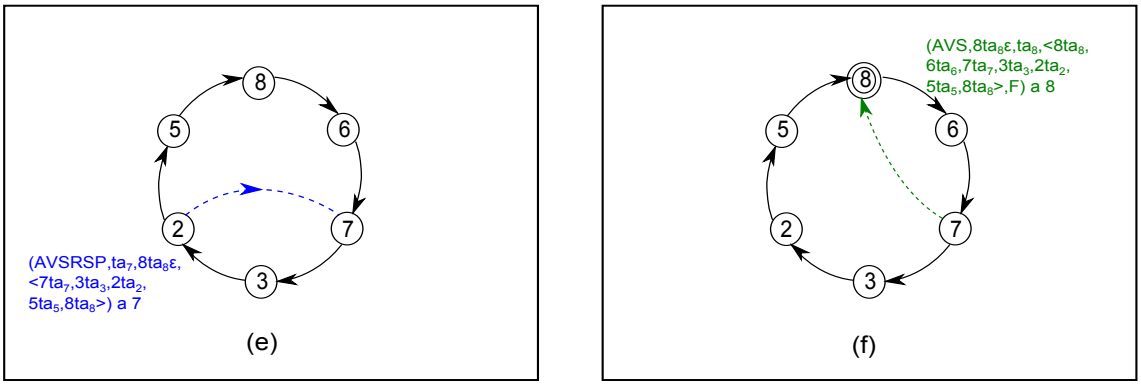

Figura 5.8: Detección con un mensaje *AVS*: fase de comparación de tres candidatos y detección final (escenario estático).

2 (nodo 7) hasta su candidato sucesor (nodo 8). El tipo de mensaje elegido para entregar toda la información recopilada por el nodo 2 es un mensaje *AVSRSP*. Para decidirse a enviar un mensaje *AVSRSP*, el nodo 2 debe establecer una nueva comparación de identidades. En esta comparativa intervienen el candidato sucesor del nodo 2, $cand\_succ_2$, y el elemento $m.sid$ almacenado en $set\_st\_avs_2$. Como $cand\_succ_2 > m_{AVS}.sid$, $(8ta_8\varepsilon > 7ta_7\varepsilon)$, el nodo 2 manda un mensaje *AVSRSP* (flecha azul de trazo discontinuo en la figura 5.8.e). Si la identidad del candidato sucesor del nodo 2 hubiera sido inferior a la del candidato predecesor conocido, el mensaje encargado de transmitir la información del nodo 2 habría sido un mensaje *AVS*. La acción $sndAVSRSP_2$ pone el siguiente mensaje en el canal de comunicaciones: (AVSRSP, ta_7, $8ta_8\varepsilon$, $<7ta_7,3ta_3,2ta_2,5ta_5,8ta_8>$). El mensaje almacenado en $set\_st\_avs_2$ no se elimina después de usarlo porque, en escenarios dinámicos, reutilizar este mensaje supone una ventaja para el rendimiento del algoritmo.

La recepción del mensaje *AVSRSP* hace que el nodo 7 sea ahora el candidato con suficiente información para tomar nuevas decisiones en el proceso de detección del interbloqueo. Sólo se procesará el mensaje si el tiempo de activación del nodo 7 coincide con el que está incluido en el mensaje, $m.ta$, lo cual es cierto por tratarse

de un escenario estático. En casos dinámicos, sin embargo, esta validación del tiempo del nodo receptor no se cumple siempre y eso permite desechar mensajes desfasados o reemplazar mensajes antiguos. Además de guardar el mensaje *AVSRSP* del nodo 2 en st_avsrsp_7, un efecto importante de la acción $rcvAVSRSP_7(2,m)$ es la actualización de la identidad del candidato sucesor. A partir de este instante de la ejecución, el nodo 7 va a considerar al nodo 8 como su candidato sucesor, $cand\_succ_7 = 8ta_8\varepsilon$, porque éste era el candidato sucesor del nodo 2. De ahí que, al comparar $cand\_succ_7$ y el elemento *m.sid* del mensaje almacenado en $set\_st\_avs_7$, ($8ta_8\varepsilon$, ta_7, $<8ta_8,6ta_6,7ta_7>$, F), el nodo 7 forme un mensaje *AVS*, dirigido al nodo 8, con la estructura indicada en la acción $sndAVS_7$. El envío del mensaje (AVS, $8ta_8\varepsilon$, ta_8, $<8ta_8,6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8>$, F) se ve reflejado en la figura 5.8.f.

La fase de detección del interbloqueo coincide con la recepción, por parte del nodo 8, del mensaje *AVS* procedente del nodo 7. El efecto de la acción $rcvAVS_8(7,m)$ que señala como víctima al nodo 8 está condicionado por la coincidencia de la identidad (simulada) del nodo 8 con la componente *m.sid* del mensaje *AVS*. Al mismo tiempo, el identificador del primer elemento de la ruta de este mensaje tiene que ser el mismo que el identificador del últitmo elemento para reconocer la existencia de un ciclo en el sistema. Como todos estos requisitos se cumplen en el mensaje (AVS, $8ta_8\varepsilon$, ta_8, $<8ta_8,6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8>$) que alcanza el nodo 8, sin necesidad de almacenarlo en $set\_st\_avs_8$, se concluye directamente que el nodo 8 ha detectado el interbloqueo. Al cambiar su estado a víctima, queda habilitada la acción $Abort_8$ para romper el ciclo (fase de resolución). Sabiendo que el nodo 8 está predestinado a ser un nodo en estado *aborted* sin posibilidad de reincorporarse al sistema activamente, variables asociadas al nodo 8 como: t_unk_8, st_alg_8, inf_need_8 y $cand\_succ_8$ vuelven a su estado inicial para impedir situaciones inconsistentes en evoluciones dinámicas que se estudiarán en los siguientes ejemplos.

Instancia 1-nodo 8	**Instancia 2-nodo 2**	**Instancia 3-nodo 7**
initiate_8	initiate_2	initiate_7
$\text{rcvALG}_5(8,m)$ $\text{rcvALG}_2(5,m)$ firstAVS_8	$\text{rcvALG}_3(2,m)$ $\text{rcvALG}_7(3,m)$ $\text{rcvAVS}_2(7,m)$ sndAVSRSP_2	$\text{rcvALG}_6(7,m)$ $\text{rcvALG}_8(6,m)$ firstAVS_7 $\text{rcvAVS}_7(8,m)$ $\text{rcvAVSRSP}_7(2,m)$ $\text{sndAVS}_7(8)$
$\text{rcvAVS}_8(7,m)$		
Abort_8		

Tabla 5.2: Secuencia de ejecución para la detección de un ciclo mediante mensaje *AVS* (escenario estático).

La tabla 5.2 resume brevemente las acciones del algoritmo y el orden establecido en la ejecución descrita. Las acciones se han agrupado por instancias, bien porque los nodos iniciadores ejecutan la acción, o bien porque se propaga su identidad (simulada). Se podría proponer otra secuenciación de acciones, pero considerándola, los mensajes generados para la detección del ciclo seguirían siendo los mismos. El orden en el que los iniciadores dan por terminadas sus expectativas de convertirse en el único candidato que detecta el ciclo está totalmente predeterminado por la disposición de los iniciadores en el ciclo. Se recomienda observar los efectos de estas acciones fijándose en los mensajes que aparecen de forma conjunta en la figura 5.9.

En el diagrama de instancias del segundo caso estático de detección, los nodos que lanzan una instancia de ejecución del algoritmo son tres. La figura 5.10 representa la evolución de las instancias iniciadas por los nodos 8, nodo 7 y nodo 2. La instancia del nodo 2 concluye en primer lugar. Gracias al intercambio de mensajes con los candidatos que le suceden y le preceden, el nodo 2 queda en condiciones de comparar las identidades (simuladas) de esos candidatos con la suya propia. Enviando un mensaje *AVSRSP* al nodo 7, el nodo 2 da por terminada la ejecución de su instancia. De los tres iniciadores involucrados en la comparación, el nodo 2 es el de menor identidad

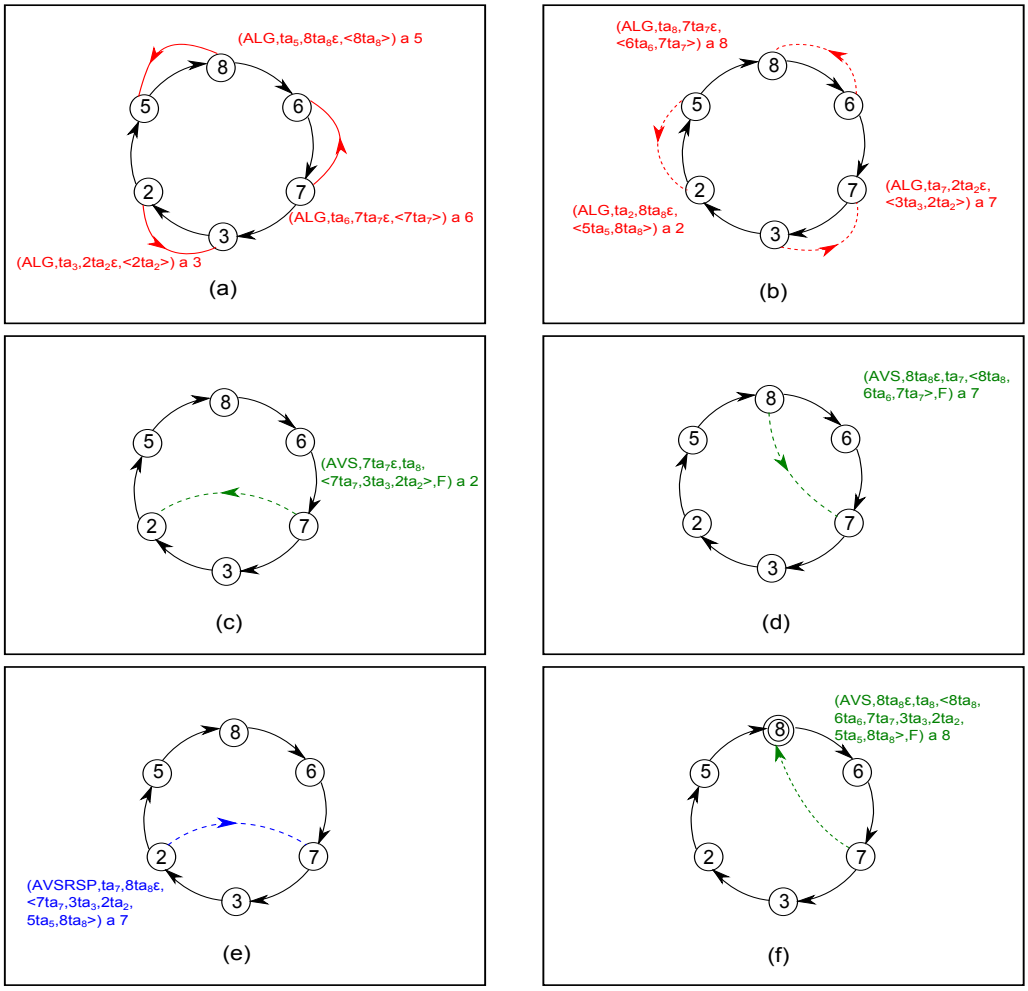

Figura 5.9: Detección con un mensaje *AVS* (escenario estático).

(simulada) y el que conoce la identidad del iniciador que debería convertirse en detector, si no hubiera más iniciadores en el sistema. Con el mensaje *AVSRSP*, el nodo 2 transfiere toda esta información al nodo 7 (flecha discontinua azul del nodo 2 al 7 en el diagrama de la figura 5.10). Como la identidad (simulada) del candidato sucesor del nodo 7 coincide con la identidad del potencial detector del ciclo que le ha apuntado el nodo 2, la instancia del nodo 7 debe también extinguirse. Para ello organiza la información recopilada en su instancia y la cedida por la instancia del nodo 2 en un mensaje *AVS*. Este mensaje *AVS* es el que representa el trasvase de información entre

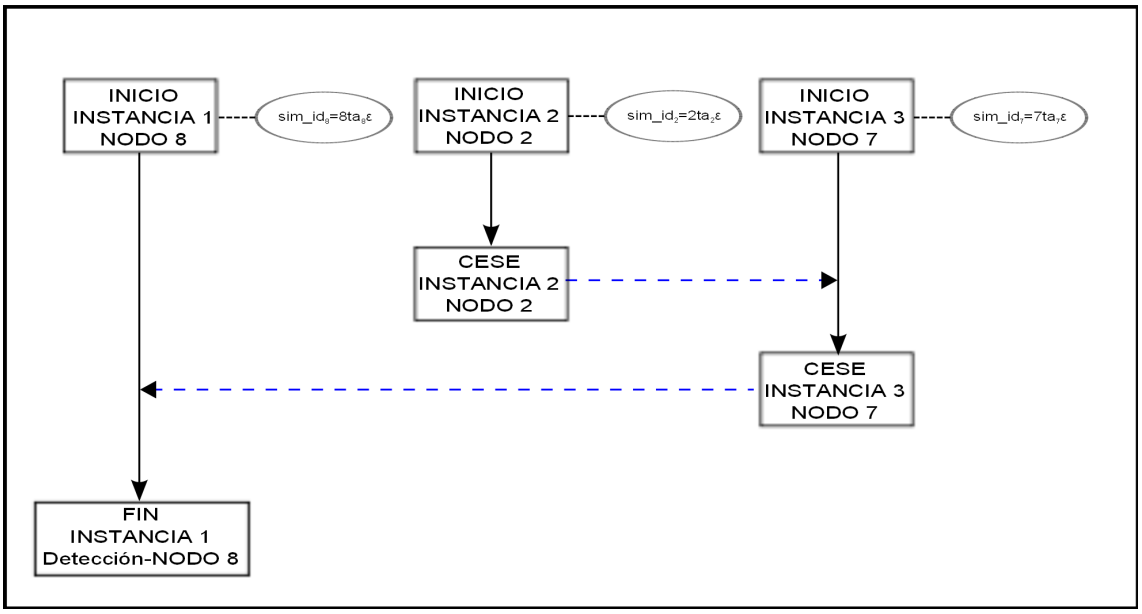

Figura 5.10: Diagrama de instancias del ejemplo de detección con mensaje *AVS* (caso estático).

la instancia del nodo 7 que termina en ese momento (flecha discontinua azul del nodo 7 al 8) y la instancia del nodo 8 (única instancia que sigue activa). Cuando concluye la instancia del nodo 8 se entiende que se ha efectuado la detección y resolución del ciclo.

5.2. Escenarios dinámicos

Los ejemplos de ejecución que se describen en los siguientes apartados se caracterizan porque la formación del ciclo que se pretende detectar es el resultado de la evolución de una configuración de nodos, en la que se modifican las relaciones de espera entre ellos. En el caso de que durante el proceso de formación del ciclo ningún nodo hubiera iniciado una instancia del algoritmo, se podría considerar que la detección del ciclo se realiza en un escenario estático. En cambio, si desaparece un nodo que inicia el algoritmo y se da a conocer como candidato detector a nodos que pertenecen

al ciclo final, la detección de ese interbloqueo se produce en un escenario dinámico. Se ha comprobado que en escenarios estáticos se recoge información correcta, pero el dinamismo la hace incorrecta. El algoritmo propuesto tiene implementados diversos mecanismos que le sirven para recomponer la información difundida a la vez que el sistema va cambiando y así, finalmente, detectar un ciclo que exista en el sistema. Con objeto de explicar todos estos procedimientos, se van a analizar seis casos diferentes que recogen situaciones en las que estos procedimientos intervienen. En esas ejecuciones se prestará especial atención a las funcionalidades que permiten propagar la identidad simulada de un nodo, reutilizar la información recopilada de configuraciones anteriores que aún tiene vigencia en la configuración actual y, por último, y no por eso menos importante, eliminar la informaciíon que no sea válida.

5.2.1. Detección de un ciclo mediante un mensaje *ALG*

Esta detección se plantea en un escenario dinámico en el que sólo un nodo lanza una instancia del algoritmo. Transcurrido un tiempo en el que esa instancia sigue su curso, el nodo iniciador se activa y sus predecesores dejan de estar bloqueados por él. La formación de un ciclo en el que están presentes nodos que colaboraron en la única instancia activa del algoritmo, pero en el que no existe ningún candidato a detector, hace que sea necesario nombrar a uno. La forma de elegir a este candidato y la identidad que debe adoptar constituyen el mecanismo fundamental para que se pueda detectar con éxito el interbloqueo en un escenario dinámico cualquiera. En definitiva, haciendo uso del caso de detección más simple en un entorno dinámico se va a describir el **mecanismo de propagación y adquisición de una identidad simulada**.

Dada la cadena de esperas representada en la figura 5.11.a, se supone que el nodo 4 inicia el algoritmo ejecutando la acción $initiate_4$. Del mismo modo que en las secciones 5.1.1 y 5.1.2, el mensaje ALG inicial del nodo 4 es redirigido a través de

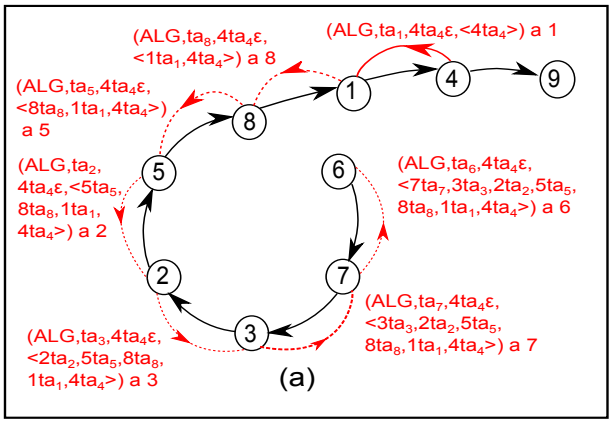

Figura 5.11: Detección con un mensaje ALG: fase de iniciación y búsqueda de candidatos (escenario dinámico).

los nodos bloqueados de la cadena hasta alcanzar el nodo 6. Todos los mensajes ALG que surgen en esta fase de búsqueda de otro candidato incluyen en la componente *sid* la identidad simulada del nodo 4, $4ta_4\varepsilon$. Esto quiere decir los nodos de la cadena que preceden al nodo 4 lo reconocen como único candidato a detector. La ruta que se almacena en st_alg_6, variable correspondiente al último nodo de la cadena, contiene los pares $<6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8,1ta_1,4ta_4>$.

Una vez que el nodo 4 ha enviado el mensaje ALG, la espera que representa su bloqueo por el nodo 9 puede desaparecer mediante la ejecución de $StartDelArc_9(4,ta_4)$ y $EndDelArc_4(9)$. Hay que tener en cuenta que, si se desea que el mensaje ALG inicial del nodo 4 llegue hasta el nodo final de la cadena, el borrado de esperas no debe adelantarse a la ejecución de las acciones $rcvALG_i(j,m)$. Si esto sucediera, la redirección del mensaje ALG se truncaría porque la condición temporal que se precisa para su tratamiento no podría cumplirse. Uno de los efectos de la acción $EndDelArc_i(j)$ consiste en incrementar el tiempo de activación del nodo que se activa, el nodo i. Por tanto, al compararse el tiempo del nodo receptor del mensaje ALG y el tiempo asociado al destino que aparece en el mensaje y no coincidir, el mensaje ALG sería retirado del canal pero no sería procesado convenientemente.

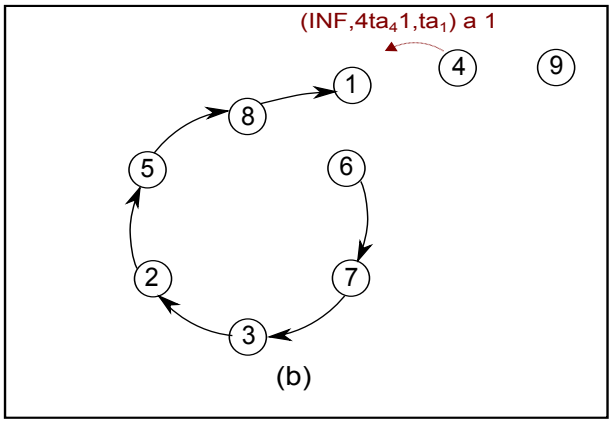

Figura 5.12: Detección con un mensaje ALG: propagación de la identidad simulada del iniciador del sistema (escenario dinámico).

Siguiendo con el proceso que modifica las esperas de la cadena de nodos, la siguiente activación que debe realizarse es la del nodo 1. Cuando se inicia el borrado de la espera del bloqueo del nodo 1 por el nodo 4, $StartDelArc_4(1,ta_1)$, surge en el sistema un nuevo mensaje. Este mensaje se denomina mensaje *INF* y es el encargado de propagar información relacionada con la identidad (simulada) de un nodo que queda activo y sin predecesores, después de haber iniciado la ejecución del algoritmo. El mensaje *INF* se envía a los nodos que recibieron un mensaje ALG, por lo que recorre el grafo de esperas en el mismo sentido que lo hizo el mensaje ALG inicial (en este ejemplo el mensaje *INF* viaja del nodo 4 al nodo 1, ver figura 5.12.b). En la representación gráfica empleada un mensaje *INF* se corresponde con una flecha de color marrón con trazo de puntos y punta triangular. Aunque se ha mencionado que el mensaje contiene la identidad simulada del nodo 4, la información que se incluye no es exactamente esa. En el tercer campo de $m_{INF}.sid$ aparece una cadena numérica de valor 1. Este valor representa la cadena de propagación de la identidad simulada y, en este caso, simboliza que una espera por la que se difundió la identidad del nodo 4 ha sido eliminada. Otra componente del mensaje *INF* es el tiempo de activación del nodo al que va dirigido el nodo. La referencia temporal de los mensajes en escenarios

cambiantes comienza a tener importancia porque gracias a ella se pueden rechazar mensajes no válidos que, siendo tratados, conducirían a detecciones falsas.

Para alcanzar la configuración de nodos registrada en la figura 5.13.c, debe procederse al borrado de la espera entre el nodo 1 y el nodo 8. La eliminación de esta espera requiere haber ejecutado previamente la acción $EndDelArc_1(4)$. En cuanto el nodo 1 se convierte en un nodo activo por los efectos de $EndDelArc_1(4)$, la variable $status.id_1$ pasa a tomar el valor *unknown*. En el estado *active-unknown*, un nodo espera adquirir una nueva identidad simulada. Todo nodo bloqueado, como el nodo 1 de este ejemplo, que se convirtió en un nodo *dummy* al paso de un mensaje ALG, al activarse queda a la espera de recibir un mensaje de tipo INF que le transfiera una identidad simulada. Adquiriendo esa identidad simulada podrá hacer las veces del candidato detector (nodo 4) que conoció en la configuración de nodos donde él sigue participando.

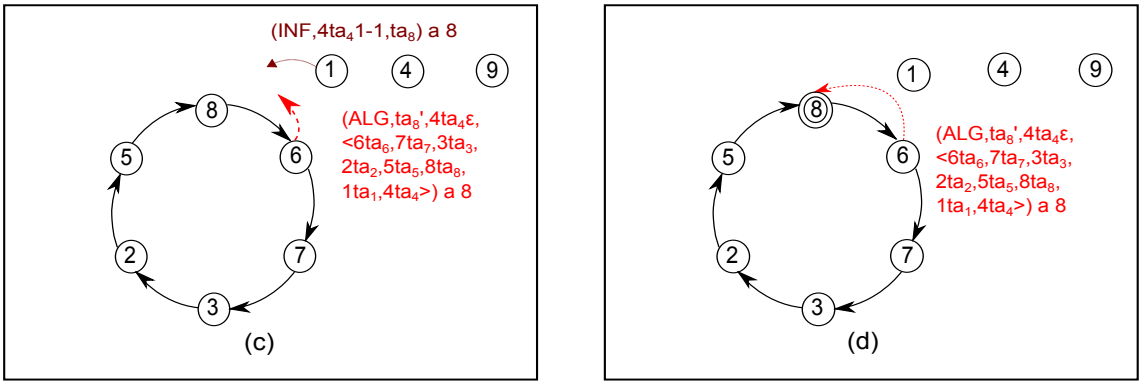

Figura 5.13: Detección con un mensaje ALG: adquisición de la identidad simulada de un iniciador y detección final (escenario dinámico).

Suponiendo que se retrasa la recepción del mensaje INF y la modificación del grafo de esperas avanza, se llega a una situación en la que no surge ningún mensaje entre los nodos. Al eliminar completamente la espera del nodo 1 al nodo 8 mediante las acciones $StartDelArc_1(8,ta_8)$ y $EndDelArc_8(1)$, el nodo 8 también pasa a estado

active-unknown. El nodo 8 no conseguirá su identidad simulada hasta que el nodo 1 la reciba previamente y forme un nuevo mensaje para propagarla hasta el nodo 8. A pesar de que el nodo 8 ha quedado a la espera de una nueva identidad simulada, el medio no está limitado y puede que haya más modificaciones en el grafo de esperas. Concretamente, el nodo 8 puede bloquearse de nuevo, esta vez por el nodo 6 ejecutándose la acción $StartAddArc_8(6)$. Al mismo tiempo que esta espera termina de formarse, los efectos de la acción $EndAddArc_6(8,ta_8\prime)$ hacen que el nodo 6 envíe un mensaje ALG con el contenido de su almacén st_alg_6. El mensaje ALG no será retirado del canal por el nodo 8 hasta que $status.id_8$ sea *known*, o sea, hasta que no haya procesado convenientemente el mensaje INF que el nodo 1 debe mandarle para actualizar su identidad simulada y evitar inconsistencias.

En este punto en el que no hay más acciones habilitadas, se obliga a que el nodo 1 ejecute la acción $rcvINF_1(4,m)$. Considerando que antes de activarse el nodo 1 era un nodo *dummy*, la identidad simulada se adquiere directamente del valor del mensaje, $sim\_id_1 = 4ta_41$. Como el nodo 1 tiene todavía almacenado en $setPredToInf_1$ la identidad del nodo 8, otro de los efectos de la acción $rcvINF_1(4,m)$ provoca que se forme un nuevo mensaje INF con destino el nodo 8. La identidad simulada que se incluye en este mensaje se construye a partir de la que acaba de adquirir el nodo 1 modificando su tercer campo. La longitud de la cadena de propagación de la identidad simulada debe incrementarse ya que el mensaje INF va a superar otra espera eliminada. Por otro lado, la parte de la identidad simulada correspondiente al identificador y al tiempo conservan su contenido. De acuerdo con todo esto, la identidad que viajará en el mensaje INF es $4ta_41$-1. Después de haberse puesto en marcha la generación de mensajes INF con la identidad simulada adecuada para los predecesores del nodo 1, éste cambia su estado a *active-known*. En ese estado, el nodo se ha desligado completamente de la configuración de nodos y vuelve a cambiar de identidad simulada. Para evoluciones posteriores, el nodo 1 actuará con $sim\_id_1 = 1ta_1\prime\varepsilon$.

La siguiente acción que se lleva a cabo es la recepción del mensaje *INF* por parte del nodo 8. Aunque el nodo 8 está bloqueado por el nodo 6, la variable $status.id_8$ es *unknown*. Ésta, junto con la condición temporal que confirma que el mensaje *INF* contiene información útil para el nodo 8, son las precondiciones que permiten que el nodo 8 asuma una nueva identidad simulada. Como el nodo 8, antes de activarse, también fue un nodo *dummy*, la identidad simulada del mensaje *INF* se almacena directamente, $sim_id_8 = 4ta_4$1-1. En este caso, no se crean nuevos mensajes *INF* porque los predecesores del nodo 8 no han modificado su relación de espera desde que se bloquearan por él en la cadena de nodos de partida del ejemplo. Un efecto de la acción $rcvINF_8(1,m)$ que debe ser mencionado, para poder entender el papel que asume el nodo 8 después de adquirir una nueva identidad y $status.id_8 = known$, es el cambio de estado que experimenta. Concluida la recepción del mensaje *INF*, $status.alg_8$ pasa a ser *candidate*, es decir, automáticamente el nodo 8 se convierte en un posible detector de interbloqueos y cuenta con una identidad simulada. La razón de que el nodo 8 no mantenga el estado anterior a la ejecución de $rcvINF_8(1,m)$, $status.alg_8 = blocked$, es que su comportamiento se asimila al de un nodo bloqueado que inicia el algoritmo.

Siendo $status.id_8 = known$, ya puede ser retirado del canal de comunicaciones el mensaje ALG que quedó pendiente de ser recibido. Este mensaje ALG sirve para detectar el interbloqueo que se formó al bloquearse el nodo 8 por el nodo 6 (véase figura 5.13.d). Al analizar la ruta de este mensaje se observa que contiene algunos elementos que no pertenecen al ciclo. De la ruta $<6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8,1ta_1,4ta_4>$, los dos últimos pares están completamente desligados de la configuración final. La presencia del nodo 1 y el nodo 4 no impide que la detección del interbloqueo se realice con éxito porque el algoritmo busca al receptor del mensaje ALG, nodo 8, entre los identificadores de todos los pares. Como el nodo 1 y el nodo 4 están a la derecha del nodo 8, se considera que esa parte de la ruta tiene información desfasada. Esos nodos quedaron

registrados en la ruta antes de que se eliminaran las esperas correspondientes de la cadena inicial. Por otra lado, aunque el nodo 8 pertenece al ciclo, el par de la ruta $8ta_8$ cuenta con un tiempo que no coincide con el tiempo de activación actual del nodo. Comprobar que el par $5ta_5$ está incluido en $setPred_8$ es suficiente para validar toda la ruta hasta el nodo 8, incluso apareciendo con un tiempo incorrecto, puesto que el nodo 8 no lo ha liberado. De esta forma realmente se verifica que la espera del nodo 5 al nodo 8 no se ha visto modificada desde que se creó hasta el instante en que se está produciendo la detección.

Tal y como se ha puesto de manifiesto en el desarrollo de este ejemplo, el mecanismo de propagación de una identidad simulada está integrado en las acciones del algoritmo: $StartDelArc_i(j,t)$ y $rcvINF_i(j,m)$. En la primera acción la propagación se produce al mismo tiempo que las esperas de la configuración desaparecen. En el caso de la segunda acción, la propagación se produce avanzado ya el proceso de borrado de esperas. Ambas acciones generan mensajes tipo *INF* para difundir la identidad simulada de un nodo iniciador. Por consiguiente, la adquisición de una identidad simulada sólo se puede realizar en la ejecución de $rcvINF_i(j,m)$.

La tabla 5.3 recoge el orden de ejecución de las acciones necesarias para detectar el interbloqueo final. El ciclo se ha formado al dejar evolucionar dinámicamente al sistema. Las acciones que tienen que ver con el borrado de esperas y la aparición de nuevas esperas aparecen en la columna denominada dinamismo. En esa misma columna aparecen también las acciones que desarrollan el mecanismo de propagación y adquisición de identidades simuladas. Con este mecanismo se consigue adaptar un entorno cambiante a uno estático. Para asociar las acciones con los mensajes que intervienen es conveniente disponer de una gráfica conjunta de todo el proceso (ver figura 5.14).

Instancia 1-nodo 4	**Dinamismo**	
initiate_4		
$\text{rcvALG}_1(4, m)$ $\text{rcvALG}_8(1, m)$ $\text{rcvALG}_5(8, m)$ $\text{rcvALG}_2(5, m)$ $\text{rcvALG}_3(2, m)$ $\text{rcvALG}_7(3, m)$ $\text{rcvALG}_6(7, m)$		
	$\text{SDA}_9(4, ta_4)$ $\text{EDA}_4(9)$ $\text{SDA}_4(1, ta_1)$ $\text{EDA}_1(4)$ $\text{SDA}_1(8, ta_8)$ $\text{EDA}_8(1)$ $\text{SAA}_8(6)$ $\text{EAA}_6(8, ta_8\prime)$ $\text{rcvINF}_1(4, m)$	
	$\text{rcvINF}_8(1, m)$	**Instancia 1(*)-nodo 8**
		$\text{rcvALG}_8(6, m)$
		Abort_8

Tabla 5.3: Secuencia de ejecución para la detección de un ciclo mediante mensaje *ALG* (escenario dinámico).

En el esquema que aparece en la figura 5.15 se observa cómo el nodo 4 inicia el algoritmo. Esa instancia progresa hasta que cede el testigo al nodo 8. Aunque parece que el nodo 8 comienza una nueva instancia, realmente sigue la ejecución de la instancia iniciada por el nodo 4 (tercera columna de la tabla 5.3). Esto se debe a que el nodo 8, al asumir la identidad simulada procedente del nodo 4, maneja toda la información que le ha traspasado y utiliza su nueva identidad para la toma de decisiones. El detector de un ciclo que se ha formado como resultado de una evolución dinámica es el candidato de mayor identidad simulada. El nodo 8, finalmente, detecta y resuelve el interbloqueo formado.

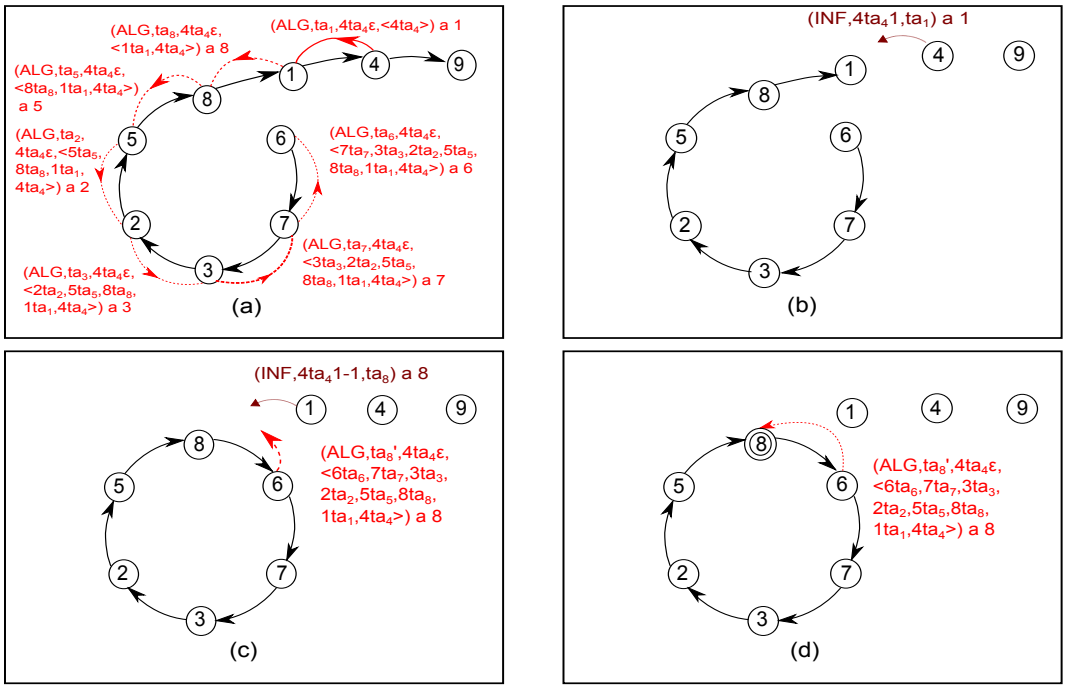

Figura 5.14: Detección con un mensaje ALG (escenario dinámico).

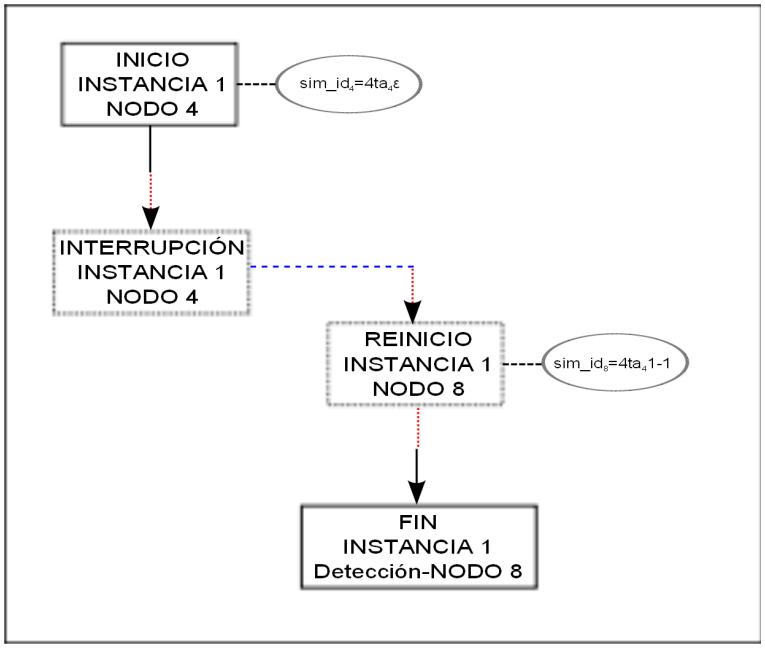

Figura 5.15: Diagrama de instancias del ejemplo de detección con ALG (caso dinámico).

5.2.2. Detección de un ciclo mediante un mensaje *AVS*

La detección de un ciclo mediante un mensaje ALG en un escenario que ha evolucionado conforme se ejecuta el algoritmo es, como ya se ha comentado en el apartado anterior, un caso sencillo que se produce cuando sólo una instancia del algoritmo permanece activa al formarse el interbloqueo. Si el número de instancias activas del algoritmo en el momento de la formación del ciclo es superior a uno, la detección de éste se realizará mediante un mensaje AVS. Este mensaje se caracteriza por contener en su ruta al nodo receptor como identificador del primer y del último de sus pares.

5.2.2.1. Detección por un nodo no iniciador que adquiere una identidad simulada

Aprovechando la descripción de este nuevo ejemplo de ejecución, se va a explicar detenidamente el **mecanismo de retroceso de mensajes *AVS***. Tras la evolución dinámica de una configuración de nodos, es posible que las variables de almacenamiento contengan rutas con referencias a nodos que no forman parte de las relaciones de espera vigentes. Si se forma un mensaje AVS a partir de esas rutas y su destino es un nodo de los que ya no tienen vinculación con las esperas analizadas, será necesario reconducirlo hasta un nodo que sí participe en la configuración de nodos considerada. La finalidad última de este mecanismo es, por tanto, salvaguardar la mayor cantidad posible de información válida, que contiene el mensaje AVS que se dirigió fuera de la configuración de nodos relacionados. La funcionalidad descrita se incluye en varias acciones del algoritmo, concretamente, en la acción $StartDelArc_i(j,t)$, en la acción $Abort_i$ y en la acción $rcvAVS_i(j,m)$. En las dos primeras acciones que se mencionan, el retroceso de mensajes AVS se emplea para rescatar información almacenada en $set\_st\_avs_i$ antes de vacíar su contenido. En lo que respecta a la acción $rcvAVS_i(j,m)$, este mecanismo impide que la información enviada a un nodo, que no forma parte ya

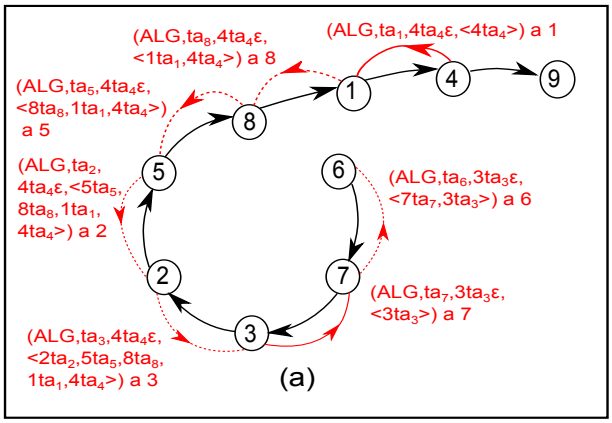

Figura 5.16: Detección de un nodo no iniciador mediante un mensaje *AVS*: fase de iniciación y búsqueda de candidatos (escenario dinámico).

del grafo de esperas, sea almacenada indebidamente y utilizada posteriormente. En la ejecución que se describe seguidamente, se muestra el mecanismo de retroceso de mensajes *AVS* asociado a la acción $rcvAVS_i(j,m)$.

La fase de iniciación de este ejemplo la constituye la puesta en marcha de dos instancias del algoritmo, una por parte del nodo 4 y la otra por el nodo 3. El mensaje *ALG* que genera el nodo 4 llega hasta el nodo 3 y queda almacenado en st_avsrsp_3. Los nodos 1, 8, 5, 2 y también el nodo 3, que inició por su parte otra instancia del algoritmo, asumen tras el paso del mensaje *ALG* que el nodo 4 es el candidato sucesor mayor. El mensaje *ALG* que surgió por la iniciación del nodo 3 alcanza al nodo 7 y el nodo 6. Los mensajes *ALG* iniciales y redirigidos que aparecen en la figura 5.16.a se generan y propagan de la misma forma que la descrita en los escenarios estáticos.

Después de que los distintos mensajes *ALG* han recorrido completamente la configuración de nodos de partida, se suceden una serie de modificaciones en el grafo de esperas del sistema. Los cambios en el grafo de esperas que se imponen hacen que tanto el nodo 4 como el nodo 1 se activen. Para provocar estas activaciones se deben ejecutar las acciones $StartDelArc_9(4,ta_4)$, $EndDelArc_4(9)$, $StartDelArc_4(1,ta_1)$ y $EndDelArc_1(4)$ en este orden obligatoriamente. El mecanismo de propagación de la

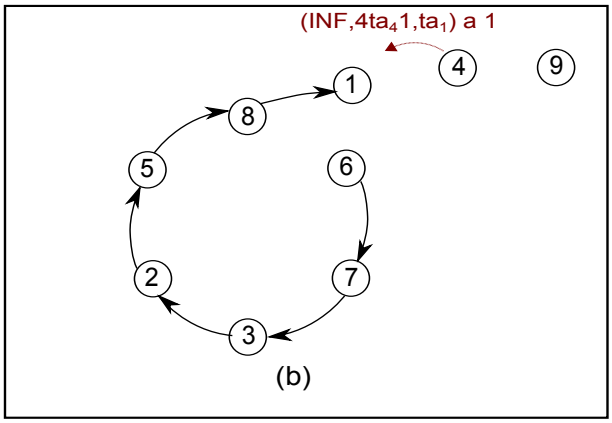

Figura 5.17: Detección de un nodo no iniciador mediante un mensaje *AVS*: propagación de la identidad simulada (escenario dinámico).

identidad que simula al nodo 4 es provocado al igual que en el apartado 5.2.1 por la acción $StartDelArc_4(1,ta_1)$. En la figura 5.17.b se observa como el nodo 4 manda un mensaje *INF* a su único predecesor (nodo 1) y le facilita la identidad $4ta_4 1$ para que simule su existencia.

El nodo 1, en estado *active-unknown* tras ejecutarse la acción $EndDelArc_1(4)$, recibe el mensaje *INF* y adquiere el valor indicado de identidad simulada. Si después de eso el nodo 1 rompe la espera que le une al nodo 8 mediante la acción $StartDelArc_1(8,ta_8)$, surge otro mensaje *INF* para el nodo 8 con la identidad simulada $4ta_4$1-1. La última componente de la identidad simulada que el nodo 1 transmite al nodo 8, se ve incrementada en un elemento, pasa de 1 a 1-1 porque registra otra espera rota de la configuración. En cuanto el nodo 8 se active ($EndDelArc_8(1)$) y pase a estado *active-unknown*, el nodo 8 cumplirá las condiciones para procesar el mensaje *INF* que le envía el nodo 1.

Estando el nodo 8 activo y pendiente de recibir el mencionado mensaje *INF*, éste puede iniciar una nueva espera. En el ejemplo, la acción $StartAddArc_8(6)$ y $EndAddArc_6(8,ta_8\prime)$ son ejecutadas para formar la espera entre los nodos 8 y 6 que, casualmente, desencadena la formación de un ciclo (véase el resultado en la figura

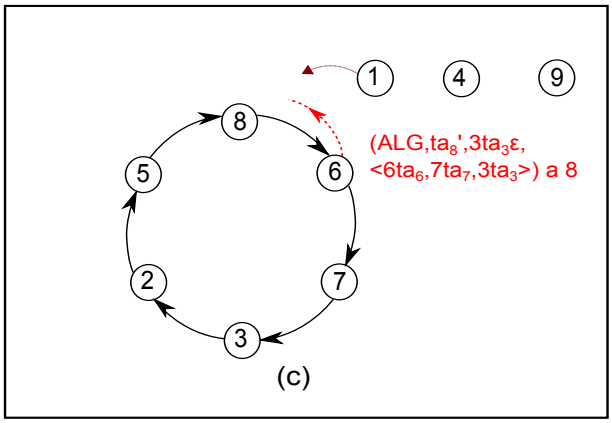

Figura 5.18: Detección de un nodo no iniciador mediante un mensaje *AVS*: búsqueda de otro candidato (escenario dinámico).

5.18.c). Sólo cuando el nodo 8 reciba el mensaje *INF*, podrá llevarse a cabo la detección de este ciclo. Por consiguiente, las modificaciones del grafo pueden continuar su curso pero las acciones propias del proceso de detección se ven suspendidas hasta que todos los nodos del ciclo pasen a estado *known*. Este aplazamiento de algunas tareas queda patente, por ejemplo, al completarse la espera del nodo 8 al nodo 6. Como consecuencia de la formación de esa espera, se genera un mensaje ALG con información del nodo 6 hacia el nodo 8. Si el nodo 8 no adquiere la identidad simulada que le corresponde, la recepción de este mensaje se verá paralizada hasta que cambie esta situación.

Finalmente, la ejecución de la acción $rcvINF_8(1,m)$ tiene lugar y el nodo 8, al adquirir la identidad simulada, participa ya plenamente en la detección del ciclo del que forma parte. Al igual que en la detección del apartado 5.2.1, el nodo 8, además de adoptar la identidad del nodo 4, pasa a ser candidato, comportándose como si él mismo hubiera sido el responsable de lanzar la instancia del algoritmo que originó el nodo 4. Para ello, el nodo 8 retira inmediatamente el siguiente mensaje ALG del canal: (ALG, $ta_8\prime$, $3ta_3$ ε, $<6ta_6,7ta_7,3ta_3>$). La información que contiene este mensaje pone de manifiesto que el nodo 3 es otro candidato en el ciclo a detectar. Esto implica

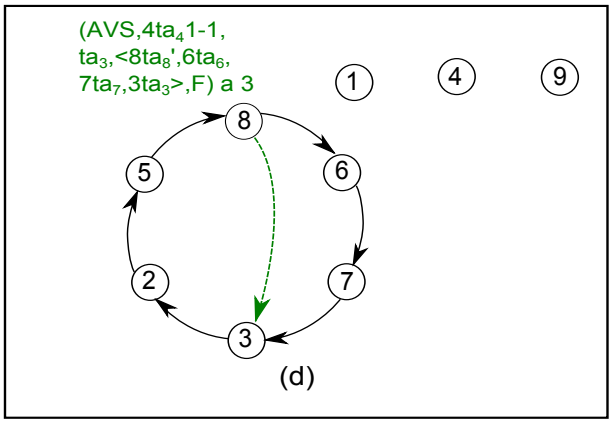

Figura 5.19: Detección de un nodo no iniciador mediante un mensaje *AVS*: comparación directa entre candidatos (escenario dinámico).

que, al ejecutar la acción $rcvALG_8(6,m)$, el nodo 8 puede comparar directamente su identidad simulada con la identidad que propaga el mensaje *ALG*. Como sim_id_8 vale $4ta_4$1-1 y $m_{ALG}.sid = 3ta_3\varepsilon$, se concluye que $sim\_id_8 > m_{ALG}.sid$. El resultado de esta comparación, tal y como se explicó ya en la sección 5.1.2, conlleva que el nodo 8 mande un mensaje *AVS* al nodo 3 indicándole que él, comportándose como el nodo 4, es el candidato mayor de los dos. Este mensaje *AVS* se genera mediante la acción $firstAVS_8$ y se observa en la figura 5.19.d.

Cuando el nodo 3 recibe ese mensaje *AVS* reconoce su inferioridad y manda como respuesta un mensaje de tipo *AVS* enlazando las rutas de las que tiene constancia. En st_avsrsp_3 está almacenada la información (ta_3, $4ta_4\varepsilon$, $<3ta_3,2ta_2,5ta_5,8ta_8,1ta_1,4ta_4>$) y el mensaje *AVS* del nodo 8 (AVS, $4ta_4$1-1, ta_3, $<8ta_8\prime,6ta_6,7ta_7,3ta_3>$, F) queda almacenado en $set\_st\_avs_3$, por efecto de la acción $rcvAVS_3(8,m)$. El mensaje resultante (AVS, $4ta_4$1-1, ta_4, $<8ta_8\prime,6ta_6,7ta_7,3ta_3,ta_2,5ta_5,8ta_8,1ta_1,4ta_4>$, F) se muestra en la figura 5.20.e. La elección del tipo de mensaje que forma el nodo 3 viene determinada por las identidades conocidas del candidato sucesor y del candidato predecesor, siendo aquí $cand\_succ_3 \leq m_{AVS}.sid$ ($4ta_4\varepsilon < 4ta_4$1-1). La comparación de estas identidades es un punto de interés en el que se evidencia que, a igual valor de identificador y

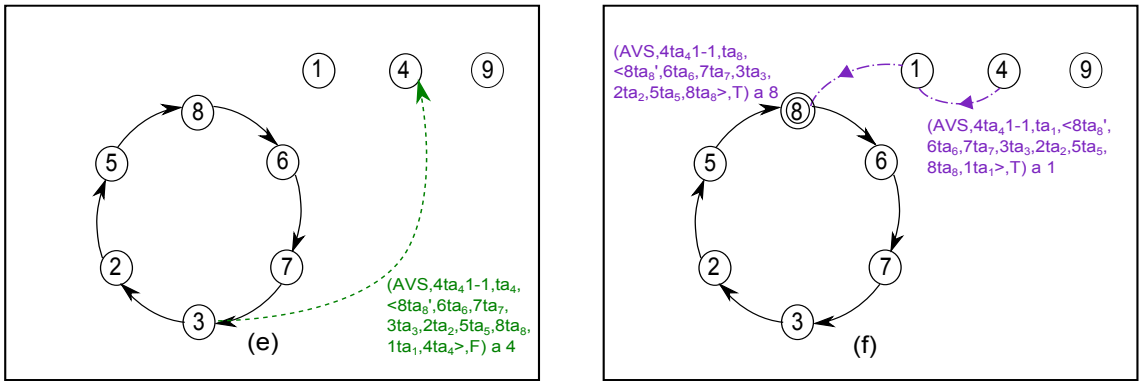

Figura 5.20: Detección de un nodo no iniciador mediante un mensaje *AVS*: retroceso de mensajes *AVS* (escenario dinámico).

de referencia temporal, la identidad simulada mayor es la que presenta la cadena de propagación de la identidad más larga. En este caso, la longitud de la cadena vacía, ε, es obviamente menor que la de la cadena 1-1.

Según se aprecia también en la figura 5.20.e el destino del mensaje *AVS* es el nodo 4. A pesar de que el nodo 4 dejó de estar relacionado con los nodos que conforman el ciclo, el nodo 3 todavía lo reconoce como su candidato sucesor mayor. Esto se debe a que la información sobre la desvinculación del nodo 4 no ha podido llegar hasta el nodo 3 porque el nodo 8 la detuvo. No hay que olvidar que el nodo 8 era un predecesor más cercano al nodo 4 y ha sido el último nodo de la configuración inicial que se ha activado. Para corregir esta situación se pone en funcionamiento el mecanismo de retroceso de mensajes *AVS* ya presentado. El objetivo de este procedimiento es conseguir redirigir el mensaje *AVS* que manda el nodo 3 al nodo 8, y no al nodo 4, sin que el nodo 4 llegue a almacenarlo. Los mensajes *AVS* en retroceso que se han diseñado para tal fin se observan en el grafo de la figura 5.20.f.

A la hora de seleccionar un destino adecuado para este tipo de mensaje *AVS*, se escoge el identificador del penúltimo par de la ruta del *AVS* que se desea dar marcha atrás. De esta manera se pone en el canal de comunicaciones un mensaje *AVS*

que, siguiendo la ruta del mensaje *AVS* con destino incorrecto, deshace el camino hecho, nodo a nodo. Conforme el mensaje *AVS* da un salto hacia atrás, la ruta que transporta también se adapta al nuevo destino, eliminando el par que se ha superado. El primer mensaje que se genera en el ejemplo que se está describiendo es (AVS, $4ta_4$1-1, ta_1, <$8ta_8$',$6ta_6$,$7ta_7$,$3ta_3$,$2ta_2$,$5ta_5$,$8ta_8$,$1ta_1$>, T). El nodo 1 recibe este mensaje y lo retransmite hacia atrás porque el nodo 1, al igual que sucedía con el nodo 4, está desligado de los nodos del ciclo. El mensaje *AVS* que rebota el nodo 1 es (AVS, $4ta_4$1-1, ta_8, <$8ta_8$',$6ta_6$,$7ta_7$,$3ta_3$,$2ta_2$,$5ta_5$,$8ta_8$>, T). Los dos mensajes *AVS* en retroceso incluyen en su campo *m.fwd* el valor booleano *T*, que distingue al mensaje *AVS* en retroceso. Tal y como se concluye, viendo la figura 5.20.f, el mensaje *AVS* que finalmente llega hasta el nodo 8 es el que permite señalarlo como víctima, para romper el interbloqueo detectado.

En la tabla 5.4 se proporciona el listado de acciones que conducen a la detección del ciclo de este ejemplo. La acciones están dispuestas según el orden de ejecución descrito anteriormente y se han agrupado de acuerdo a la instancia del algoritmo a la que están asociadas. Las acciones de la columna denominada *Dinamismo* se corresponden con los procesos de borrado y creación de esperas, así como con los mecanismos que hacen posible la adaptación del entorno dinámico al estático. Los mensajes y algunos efectos de las acciones de este ejemplo dinámico están representados de manera conjunta en la figura 5.21.

Como resumen de este ejemplo, se facilita un esquema de su ejecución en la figura 5.22. Inicialmente, se ponen en funcionamiento dos instancias del algoritmo. Una es lanzada por el nodo 4 y otra por el nodo 3. Ambas progresan en paralelo hasta que el nodo 4 pierde la gestión de su instancia. Antes de que el nodo 4 se desentienda de las tareas que conlleva la instancia que inició, éste cede toda la información recopilada al nodo 8. El traspaso de dirección entre ellos tiene lugar cuando el nodo 8 cambia su

Instancia 1-nodo 4	**Dinamismo**		**Instancia 2-nodo 3**
initiate_4			initiate_3
$\text{rcvALG}_1(4,m)$ $\text{rcvALG}_8(1,m)$ $\text{rcvALG}_5(8,m)$ $\text{rcvALG}_2(5,m)$ $\text{rcvALG}_3(2,m)$			$\text{rcvALG}_7(3,m)$ $\text{rcvALG}_6(7,m)$
	$\text{SDA}_9(4,ta_4)$ $\text{EDA}_4(9)$ $\text{SDA}_4(1,ta_1)$ $\text{EDA}_1(4)$ $\text{rcvINF}_1(4,m)$ $\text{SDA}_1(8,ta_8)$ $\text{EDA}_8(1)$		
	$\text{rcvINF}_8(1,m)$	**Instancia 1(*)-nodo 8**	
	$\text{SAA}_8(6)$ $\text{EAA}_6(8,ta_8\prime)$		
		$\text{rcvALG}_8(6,m)$ firstAVS_8	 $\text{rcvAV}_3(8,m)$ $\text{sndAV}_3(4)$
	$\text{rcvAVS}_4(3,m)$ $\text{rcvAVS}_1(4,m)$		
		$\text{rcvAVS}_8(1,m)$	
		Abort_8	

Tabla 5.4: Secuencia de ejecución para un escenario dinámico con detección de mensaje *AVS* por un nodo no iniciador.

identidad simulada por una que indica cómo se relacionaba con el nodo 4 ($sim_id_8 = 4ta_4$1-1). Usando esta identidad simulada, el nodo 8 retoma la ejecución de la instancia iniciada originalmente por el nodo 4. La comparación de las identidades simuladas del nodo 8 y del nodo 3 llevará a este último a dar por terminada su propia instancia. Para eso, el nodo 3 transfiere su información a la instancia dirigida por el nodo 8. Con toda la información que posee, el nodo 8 concluirá el proceso de detección, finalizando así la instancia de la que se hizo cargo tras el abandono del nodo 4.

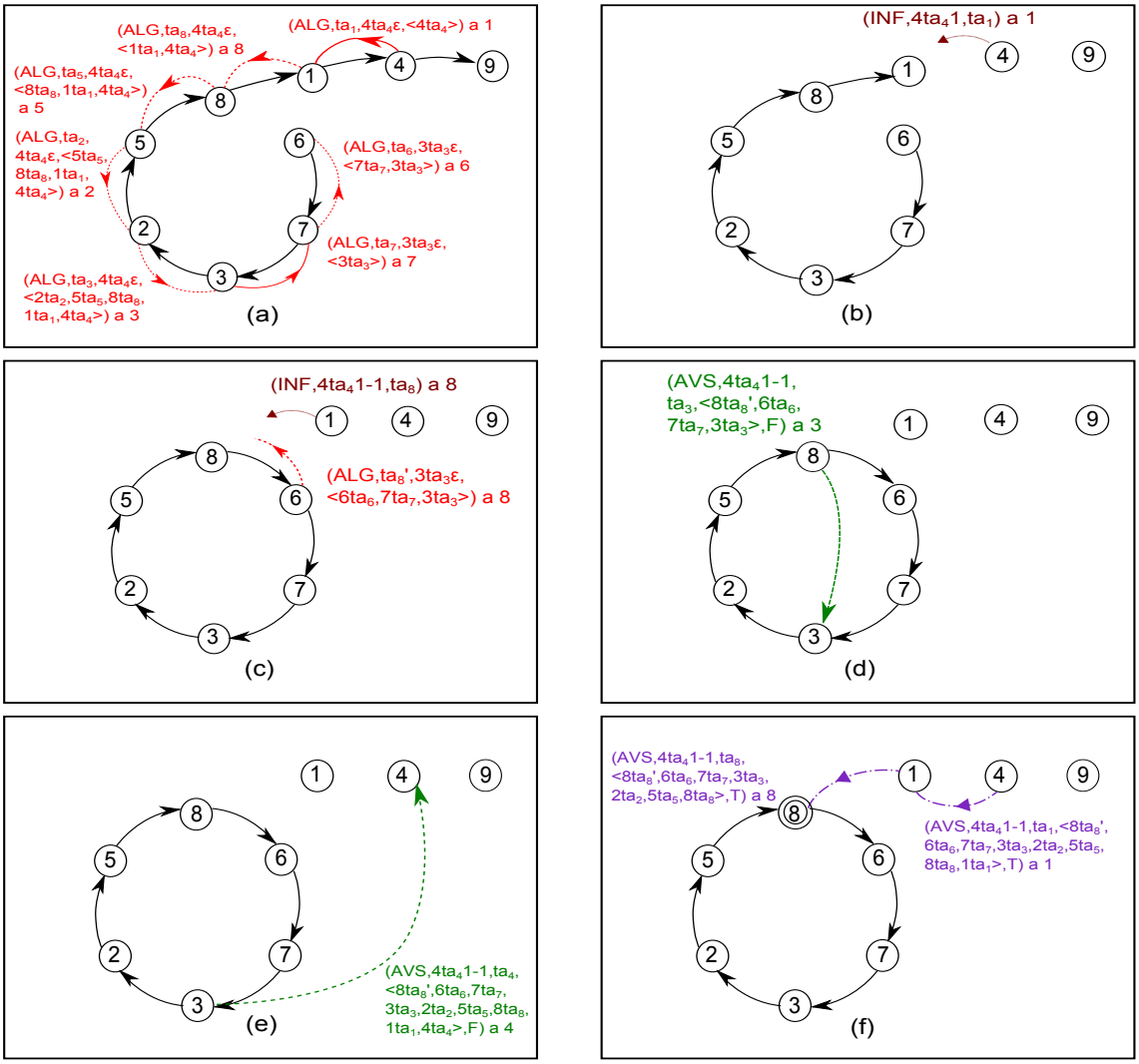

Figura 5.21: Detección de un nodo no iniciador mediante un mensaje *AVS* (escenario dinámico). Mecanismo de retroceso de mensajes *AVS*.

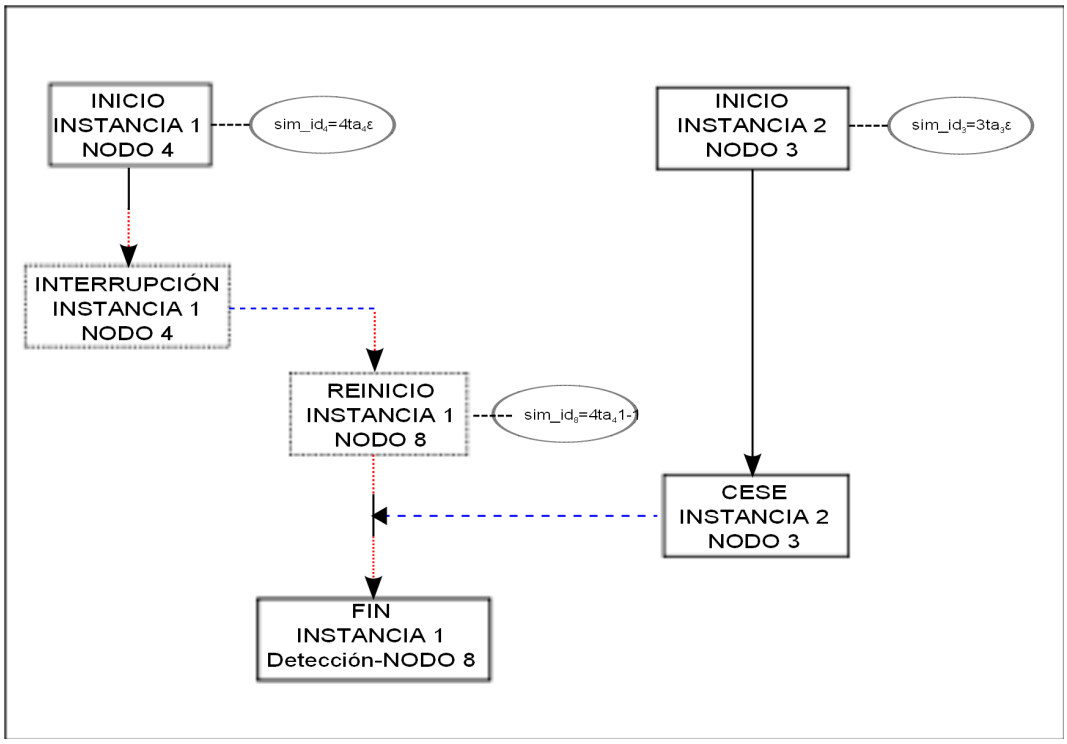

Figura 5.22: Diagrama de instancias del ejemplo de detección de un nodo no iniciador mediante un mensaje *AVS* (caso dinámico).

5.2.2.2. Detección por un nodo iniciador (*candidate*) que no adquiere una nueva identidad simulada

A simple vista este ejemplo de ejecución puede parecer similar al anterior, a no ser que se fije la atención en el hecho de que se lanzan tres instancias del algoritmo en vez de dos. La evolución dinámica del escenario planteado requiere que, en primer lugar, se ponga en funcionamiento el mecanismo de propagación y adquisición de identidades simuladas junto con el retroceso de mensajes *AVS* (efecto de la acción $StartDelArc_i(j,t)$). A diferencia de las detecciones de las secciones 5.2.1 y 5.2.2.1, el proceso de adquisición de una identidad simulada no llega a completarse y este hecho conduce a una situación que debe controlarse. El mecanismo que se va a describir en este apartado está pensado para retirar del canal de comunicaciones mensajes *INF* que

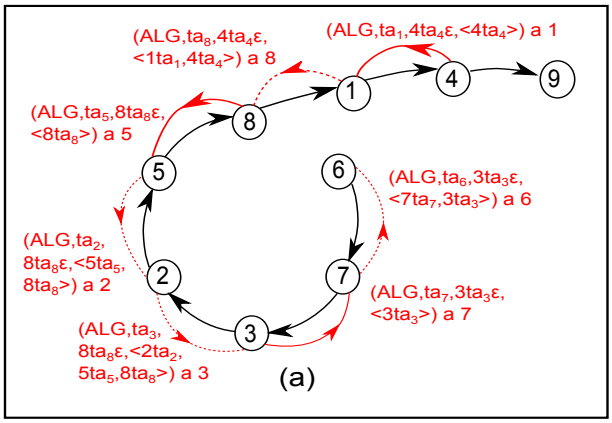

Figura 5.23: Detección de un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*: fase de iniciación y búsqueda de candidatos (escenario dinámico).

transportan identidades simuladas que, o bien no son necesarias por la configuración de nodos existente, o bien la evolución dinámica del sistema las hace inservibles. De estos dos casos en los que se emplea el **mecanismo de borrado de mensajes *INF*** se va a desarrollar el primero de ellos. Ni qué decir tiene que retirar este tipo de mensajes de los canales de comunicación resulta una práctica recomendable para que el sistema mejore su rendimiento.

Como configuración de partida, se cuenta con el grafo de esperas de la figura 5.23.a. De todos los nodos que aparecen en esa cadena de esperas, se supone que el nodo 4, el nodo 8 y el nodo 3 inician la ejecución del algoritmo y constituyen el grupo de candidatos a detector de posibles interbloqueos. La figura 5.23.a muestra, además de los mensajes ALG iniciales de los nodos iniciadores, los mensajes ALG que redirigen los nodos bloqueados hasta toparse con un nodo candidato. Por consiguiente, en la fase de búsqueda de otros candidatos, el mensaje ALG del nodo 4 se difunde hasta el nodo 8, el del nodo 8 llega hasta el nodo 3 y el del nodo 3 se propaga hasta el nodo 6. El mensaje ALG que recibe el nodo 8 se procesa según los efectos de la acción $rcvALG_8(4,m)$: se almacena el mensaje ALG en st_alg_8, se reconoce como candidato

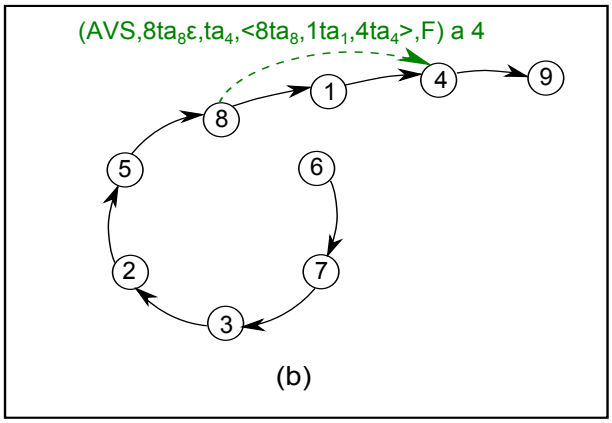

Figura 5.24: Detección de un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*: comparación de candidatos directos (escenario dinámico).

sucesor al nodo 4 y, en vista de que la identidad simulada del 8 es superior a la de su candidato sucesor ($8ta_8\varepsilon > 4ta_4\varepsilon$), se habilita la acción $firstAVS_8$. Por su parte, el nodo 3 admite la inferioridad de su identidad simulada frente a la de su candidato sucesor ($3ta_3\varepsilon < 8ta_8\varepsilon$) e incorpora sus datos al mensaje *ALG* que ha recibido para, finalmente, almacenarlo en st_avsrsp_3. Por último, el mensaje *ALG* que llega al nodo 6 se guarda en st_alg_6 para poder retransmitirlo, en caso de que un nodo se pusiera a esperar por él en el futuro.

La consecuencia de la ejecución de la acción $firstAVS_8$ es un mensaje *AVS* del nodo 8 al nodo 4. Este mensaje, que se representa en la figura 5.24.b, se almacena en $set\_st\_AVS_4$ como ($8ta_8\varepsilon$, ta_4, $<8ta_8,1ta_1,4ta_4>$, F).

Después del intercambio de mensajes descrito, el grafo de esperas sufre una serie de modificaciones. Los nodos 9, 4, 1 y 8 se activan, es decir, las relaciones de espera existentes entre ellos se cancelan. La secuencia de ejecución que transforma la configuración de nodos inicial comienza con la acción $StartDelArc_9(4,ta_4)$ y le siguen, en este orden, las acciones $EndDelArc_4(9)$, $StartDelArc_4(1,ta_1)$, $EndDelArc_1(4)$, $StartDelArc_1(8,ta_8)$ y $EndDelArc_8(1)$. Entre los efectos de $StartDelArc_4(1,ta_1)$ destaca el envío de un mensaje *INF* y de un mensaje *AVS* en retroceso al nodo 1 (ver figura

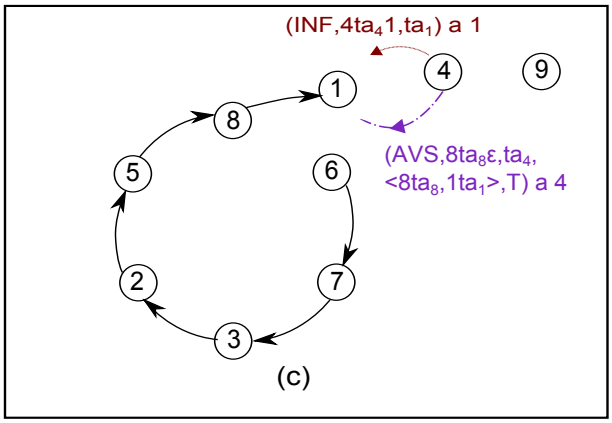

Figura 5.25: Detección de un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*: propagación de la identidad simulada y retroceso de un mensaje *AVS* (escenario dinámico).

5.25.c). El mensaje *INF* tiene como objetivo avisar de que el nodo 4 ya no es candidato a aquellos nodos que recibieron un mensaje *ALG* indicándoselo con anterioridad. En cambio, el mensaje *AVS* en retroceso aparece porque el nodo 4 se desvincula de la configuración y se inicializan todas sus variables, entre ellas, $set\_st\_avs_4$. El mecanismo de retroceso de mensajes *AVS* que tiene lugar en este ejemplo funciona de igual manera que el explicado en la sección 5.2.2.1, a pesar de que es invocado en la acción $StartDelArc_4(1,ta_1)$, en vez de en la acción $rcvAVS_i(j,m)$. La estructura del *AVS* en retroceso, (AVS, $8ta_8\varepsilon$, ta_1, $<8ta_8,1ta_1>$, T), tiene las mismas componentes que los *AVS* en retroceso creados en la citada sección.

El nodo 1 admite el mensaje *INF* siempre que se haya ejecutado previamente la acción $EndDelArc_1(4)$ y, por tanto, haya cambiado su estado a *active-unknown*. Si el nodo 1, a su vez, ya ha ejecutado la acción $StartDelArc_1(8,ta_8)$, puede enviar un mensaje *INF* al nodo 8 con la identidad simulada que le corresponde según el número de esperas rotas que le unen al nodo 4 (ver figura 5.26.d). En lo que respecta al mensaje *AVS* en retroceso, hay que mencionar que, en las mismas condiciones en las que el nodo 1 genera el nuevo mensaje *INF* para el nodo 8, el nodo 1 no crea

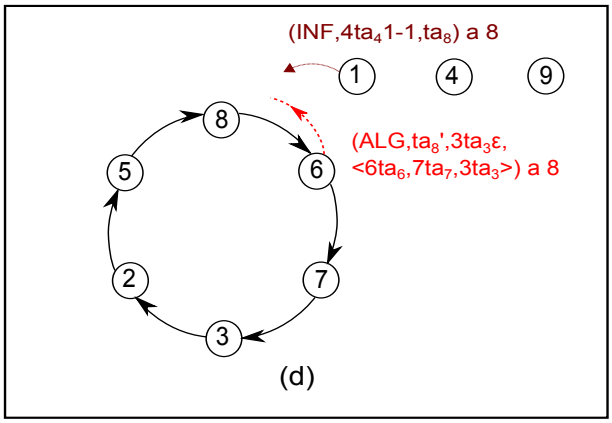

Figura 5.26: Detección de un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*: propagación de la identidad simulada y formación del ciclo (escenario dinámico).

ningún nuevo mensaje *AVS* en retroceso. Si se construyera este mensaje para el nodo 8, su ruta estaría formada por un único elemento, el par $<8ta_8>$. Parece obvio que enviar un mensaje al nodo 8 con información sólo referente a él mismo no tiene ningún sentido. Por esta razón, el nodo 1 no crea el mensaje *AVS* en retroceso para el nodo 8, tal y como podría suponerse *a priori*. En consecuencia, se puede afirmar que evitar propagar mensajes *AVS* en retroceso con sólo un elemento en su ruta implica, en cierta medida, disminuye el coste en comunicación del algoritmo.

El ciclo que aparece en la figura 5.26.d supone la ejecución previa de la acción $EndDelArc_8(1)$. Como efecto de esa acción, el nodo 8 adopta el estado *active-known*. Sabiendo que en ese estado la habilitación de la acción $rcvINF_8(1,m)$ no es posible, el mensaje *INF* procedente del nodo 1 se mantiene en el canal de comunicaciones indefinidamente. Aunque el estado del nodo 8 se transformara en *known*, el mensaje *INF* seguiría permanentemente en el canal porque las referencias temporales del nodo 8 forman parte de la precondición de la acción $rcvINF_8(1,m)$ y también cambian en el proceso de activación.

Para eliminar del canal los mensajes *INF* que nunca podrán ser procesados, se

hace uso del mecanismo de borrado de mensajes *INF*. Este mecanismo equivale a la ejecución de la acción $dltINF_i(j,m)$. El efecto de retirar el mensaje *INF* del canal se produce automáticamente si chequeando las condiciones temporales del mensaje *INF* y el tiempo de activación del nodo receptor, se comprueba que el mensaje ha perdido vigencia. En este ejemplo de detección se puede ejecutar la acción $dltINF_8(1,m)$ en cuanto el nodo 8 se activa. El mensaje *INF* del nodo 1 no es necesario para el nodo 8 porque este último era candidato en la configuración inicial. Por ser candidato en la cadena inicial de esperas frenó el mensaje *ALG* del nodo 4 (los predecesores del nodo 4 no saben de la existencia del nodo 4) y además, en la comparación directa con el nodo 4 (candidato sucesor) y con el nodo 3 (candidato predecesor), se puso de manifiesto que la identidad simulada del nodo 8 era la mayor.

Resulta conveniente advertir en este punto que no todo mensaje *INF* dirigido a un nodo candidato tiene que ser eliminado del canal sin ser procesado. Aquellos candidatos, que hayan enviado mensajes que recojan información de al menos otros dos candidatos, deben esperar obligatoriamente un mensaje *INF*. La identidad simulada que transporta el mensaje garantiza que todos los nodos involucrados en las rutas que se unieron se adaptarán correctamente a los cambios en la configuración de esperas. En el próximo ejemplo de detección se explica con más detalle el comportamiento de un candidato que, tras activarse, necesita conocer el valor de la identidad simulada que le envían otros candidatos con los que intercambió información.

La formación de la espera que simboliza el bloqueo del nodo 8 por el nodo 6 requiere ejecutar las acciones $StartAddArc_8(6)$ y $EndAddArc_6(8,ta_8\prime)$. Con la incorporación de esta espera se da por terminada la evolución dinámica del grafo y se puede retomar la detección del interbloqueo considerando un escenario estático. El efecto de la acción $StartAddArc_8(6)$, que modifica el estado del nodo 8 de *active* a *candidate*, permite que éste recupere inmediatamente la funcionalidad que había perdido al

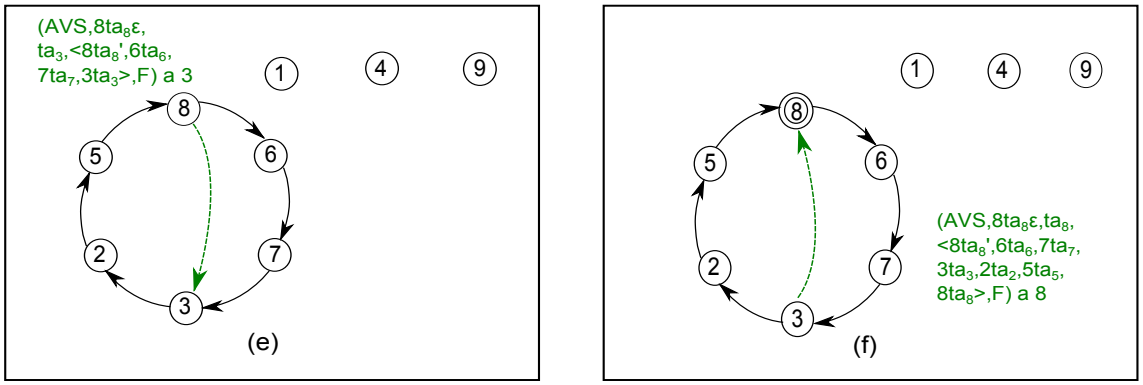

Figura 5.27: Detección de un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*: detección final (escenario dinámico).

activarse. La instancia iniciada por el nodo 8 sigue progresando como si las modificaciones del grafo de esperas no hubieran sucedido. La recepción del mensaje *ALG* por parte del nodo 8 habilita la acción $firstAVS_8$. Así que, como respuesta al mensaje *ALG* redirigido por el nodo 6, el nodo 8 manda el mensaje *AVS* que aparece en la figura 5.27.e, cuyo contenido es (AVS, $8ta_8\prime\varepsilon$, ta_3, $<8ta_8\prime,6ta_6,7ta_7,3ta_3>$, F). Cuando el nodo 3 lo retira del canal de comunicaciones, almacena el mensaje en $set\_st\_avs_3$.

Tan pronto como el nodo 3 concluye que no puede convertirse en candidato a detector, manda un mensaje *AVS*. Este mensaje organiza toda la información que el nodo 3 contiene en sus almacenes. En st_avsrsp_3 el nodo 3 guarda la ruta que llega hasta el candidato sucesor que conoce, ($8ta_8\varepsilon$, ta_3, $<3ta_3,2ta_2,5ta_5,8ta_8>$) y en $set\_st\_avs_3$, la ruta que constituye el camino que lleva hasta el candidato predecesor. Al solapar estas dos rutas por el nodo 3, que es común a ambas, resulta la siguiente ruta: $<8ta_8\prime,6ta_6,7ta_7,3ta_3,2ta_2,5ta_5,8ta_8>$. Esta ruta se incluye en un mensaje *AVS* ya que el nodo 3 comprueba que su candidato sucesor y su candidato predecesor son el mismo nodo, $8ta_8\varepsilon$. La detección final del interbloqueo tiene lugar cuando el nodo 8 retira el mensaje *AVS* del canal y se verifica que la ruta del mensaje contiene un ciclo (véase la figura 5.27.f). El hecho de que el identificador de los elementos

primero y último de la ruta no tengan asociado el mismo tiempo de activación no supone ningún inconveniente para reconocer la existencia del ciclo. El motivo de los diferentes registros temporales es que el primer y el último par de la ruta proceden de mensajes que analizaron las esperas del ciclo en distintos instantes de tiempo. Dicho de otra forma, el nodo 8 es el único nodo del ciclo que cambió su bloqueo, siguiendo vinculado al resto de nodos de la configuración inicial.

Todos los mensajes que se han representado en los grafos de la figura 5.28 surgen como efecto de la ejecución de las acciones que contiene la tabla 5.5. Se podría proponer un orden distinto para algunas de las acciones incluidas en la columna de *Dinamismo* pero, esto no supondría la aparición de nuevos mensajes o la desaparición de alguno de los ahí representados. Tanto la acción $rcvINF_1(4,m)$ como la acción $rcvAVS_1(4,m)$ podrían retrasarse hasta que finalizara la formación del ciclo, o sea, hasta la acción $EndAddArc_6(8,ta_8\prime)$. Por otro lado, la acción $dltINF_8(1,m)$ podría ejecutarse en cualquier instante posterior a la generación del mensaje m que debe eliminar, incluso después de haberse concluido la detección y resolución del interbloqueo.

El esquema de instancias que se extrae de la ejecución de este ejemplo (figura 5.29) guarda bastante parecido con el de la detección mediante mensaje *AVS* en un escenario estático (figura 5.10). Inicialmente, los nodos 4, 8 y 3 lanzan sendas instancias del algoritmo. Las tres instancias progresan en paralelo hasta que el nodo 4 abandona la dirección de su instancia. En un principio, y dado que la cadena de esperas se activa hasta el nodo 8, se procede a traspasar a éste la información que el nodo 4 recabó en su instancia. Sin embargo, este traspaso no llega a concluirse porque la instancia del nodo 8 ha transcurrido de forma independiente a la del nodo 4. La activación del nodo 8 y la incorporación de su espera por el nodo 6 suponen un paréntesis en la ejecución

de las acciones propias de la detección de interbloqueos. Cuando la evolución dinámica de las esperas finaliza y vuelve a establecerse un escenario estático, la instancia del nodo 8 retoma su ejecución normal. Esto se debe a que la identidad simulada del nodo 8 no varía mientras el grafo de esperas se modifica. El intercambio de mensajes entre los nodos que dirigen las instancias activas del sistema propicia que la instancia del nodo 3 termine. La información que cede la instancia del nodo 3 a la del nodo 8 permite, finalmente, la detección del ciclo existente en el sistema.

Instancia 1-nodo 4	**Dinamismo**	**Instancia 2-nodo 8**	**Instancia 3-nodo 3**
initiate_4		initiate_8	initiate_3
$\text{rcvALG}_1(4, m)$ $\text{rcvALG}_8(1, m)$ $\text{rcvAVS}_4(8, m)$		$\text{rcvALG}_5(8, m)$ $\text{rcvALG}_2(5, m)$ $\text{rcvALG}_3(2, m)$ firstAVS_8	$\text{rcvALG}_7(3, m)$ $\text{rcvALG}_6(7, m)$
	$\text{SDA}_9(4, ta_4)$ $\text{EDA}_4(9)$ $\text{SDA}_4(1, ta_1)$ $\text{EDA}_1(4)$ $\text{rcvINF}_1(4, m)$ $\text{rcvAVS}_1(4, m)$ $\text{SDA}_1(8, ta_8)$ $\text{EDA}_8(1)$ $\text{dltINF}_8(1, m)$ $\text{SAA}_8(6)$ $\text{EAA}_6(8, ta_8\prime)$		
		$\text{rcvALG}_8(6, m)$ firstAVS_8	 $\text{rcvAVS}_3(8, m)$ sndAVS_3
		$\text{rcvAVS}_8(3, m)$	
		Abort_8	

Tabla 5.5: Secuencia de ejecución para un escenario dinámico con detección de mensaje *AVS* por un iniciador sin adquirir identidad simulada.

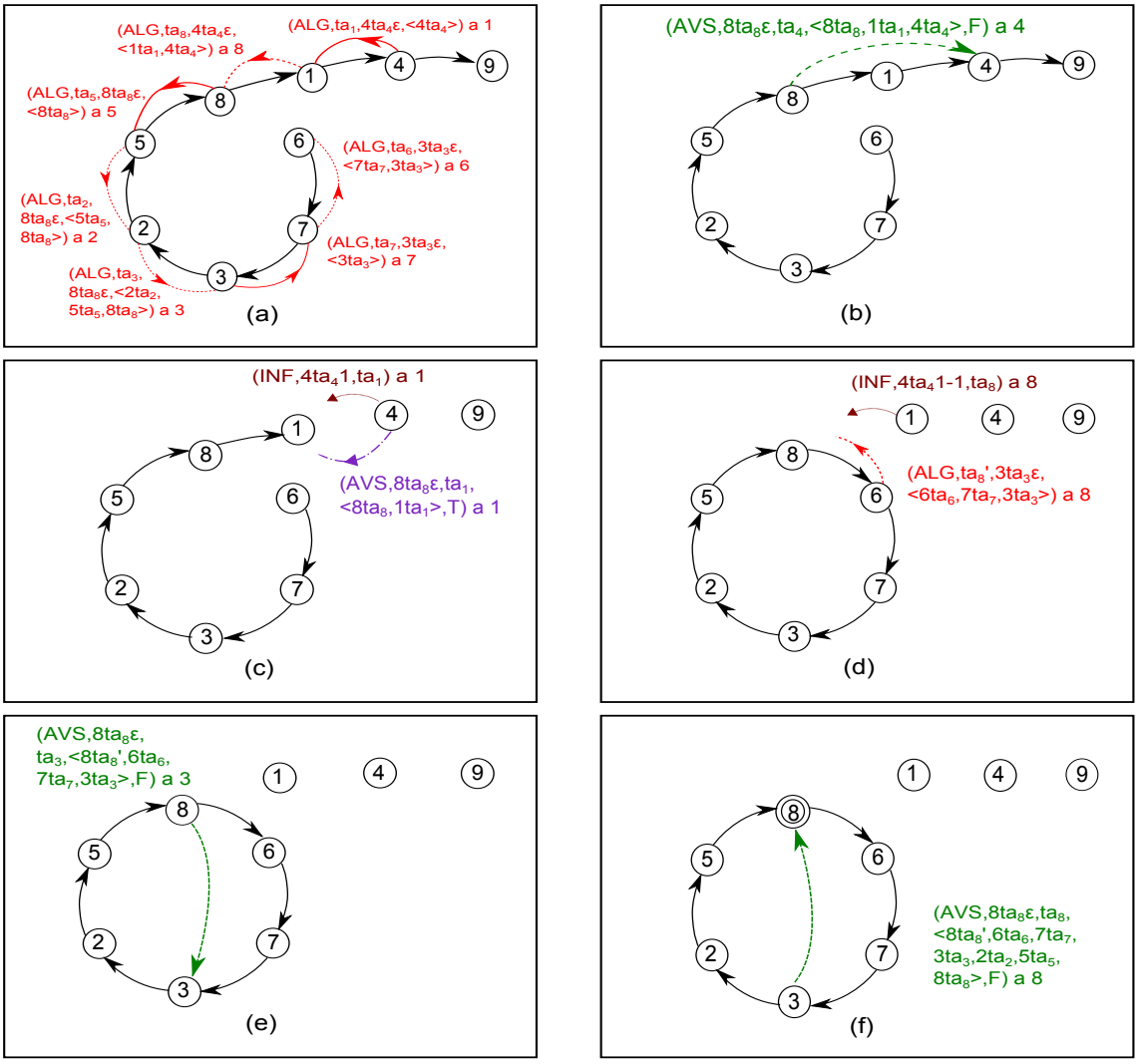

Figura 5.28: Detección por un nodo iniciador que no adquiere identidad simulada mediante un mensaje *AVS*. Retroceso de mensajes *AVS* y borrado de mensajes *INF*.

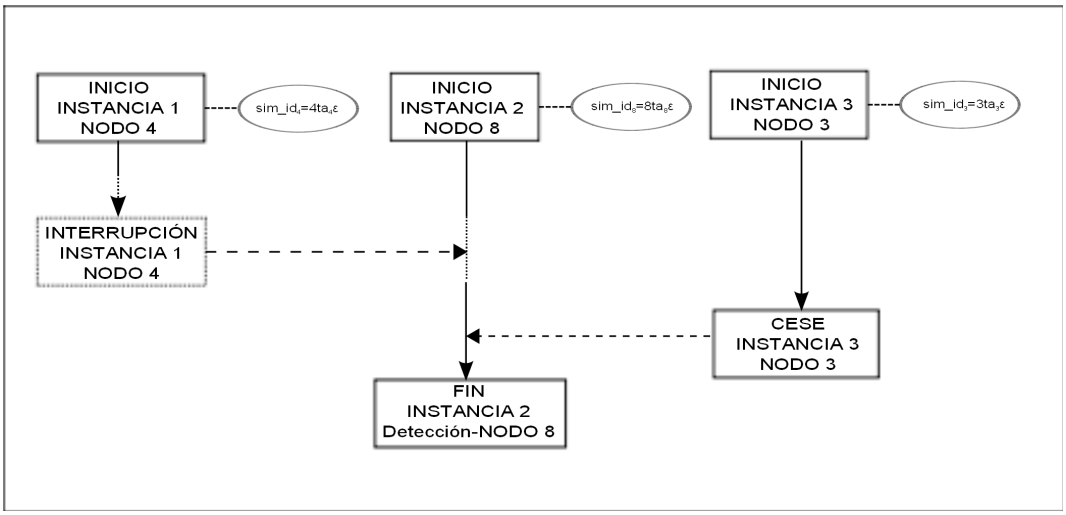

Figura 5.29: Diagrama de instancias del ejemplo de detección, por un nodo iniciador que no adquiere identidad simulada, con mensaje *AVS* (caso dinámico).

5.2.2.3. Detección por un nodo iniciador (*candidate*) que adquiere una nueva identidad simulada

El siguiente ejemplo de detección sirve para explicar el **mecanismo de reconversión de mensajes *AVSRSP***. Se trata de un procedimiento que, de nuevo, va a permitir que el sistema se adapte a los cambios que ha habido en la configuración de las esperas. Para ello se salvará la máxima información que un nodo ha recopilado en una configuración, antes de que deje de participar en ella. Con la información que se recupere y sea válida, se podrá continuar con la detección de interbloqueos sin necesidad de volver a recopilarla. En la ejecución que se muestra en este ejemplo, se rescata el contenido de la variable $set\_st\_avs_i$ antes de que el nodo i lo elimine porque adquiere una nueva identidad simulada. Al adquirir una nueva identidad, el nodo i pasa a comportarse como otro nodo y, por tanto, la información que maneja debe ser coherente con esa nueva identidad. Sólo si la nueva identidad del nodo i es superior a la identidad del candidato predecesor que conoce, se transformará el mensaje almacenado en $set\_st\_avs_i$ en un mensaje *AVSRSP* cuyo destino es precisamente ese

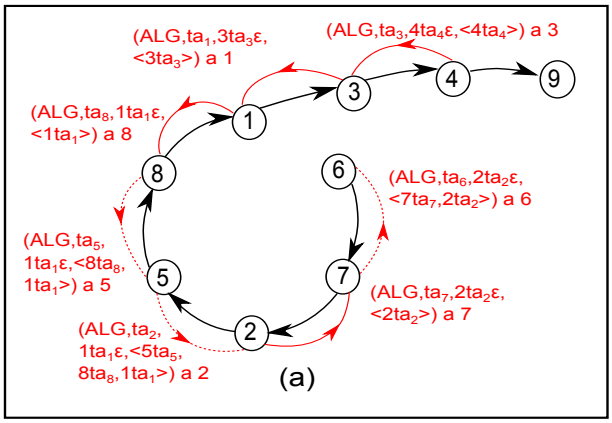

Figura 5.30: Detección por un nodo iniciador que adquiere identidad simulada mediante un mensaje *AVS*: fase de iniciación y búsqueda de otros candidatos (escenario dinámico).

candidato predecesor. Cuando se procesa este mensaje *AVSRSP*, cualquier mensaje que pudiera haber sido almacenado previamente se perderá. Esto no supone ningún problema porque la nueva información está actualizada. Además, esta información es la que se habría almacenado si el lugar del nodo i en la configuración inicial lo hubiera ocupado el nodo del que procede la nueva identidad simulada. A continuación se describe paso a paso la ejecución de un ejemplo donde se emplea el mecanismo citado.

La fase de iniciación de la detección tiene lugar cuando los nodos 1, 2, 3 y 4 inician una instancia del algoritmo cada uno. La identidad del nodo 2 se transmite hasta el nodo 6 (último nodo bloqueado de la cadena de esperas inicial) mediante mensajes *ALG*. De igual modo, la identidad del nodo 4 llega hasta el nodo 3. Como el nodo 3 es un candidato de identidad inferior al candidato 4, en st_avsrsp_3 se almacena: (ta_3, $4ta_4\varepsilon$, $<3ta_3,4ta_4>$). Al nodo 1 le ocurre algo similar y guarda en st_avsrsp_1 el mensaje *ALG* que recibió de 3, esto es, (ta_1, $3ta_3\varepsilon$, $<1ta_1,3ta_3>$). Sin embargo, el nodo 2 recibe el mensaje *ALG* del nodo 1 y se habilita el envío de un mensaje *AVS*, estableciendo que el nodo 2 es un candidato mayor que el nodo 1. Los mensajes

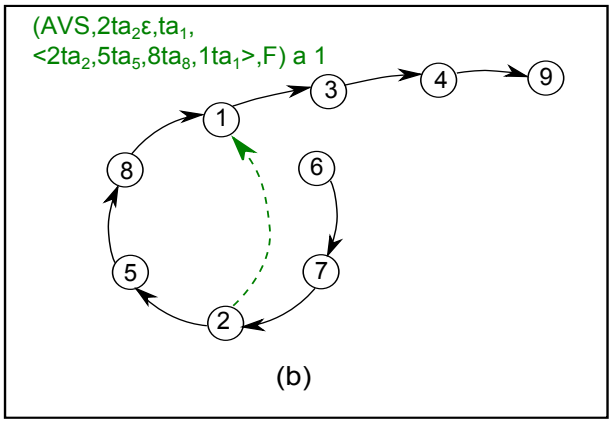

Figura 5.31: Detección por un nodo iniciador que adquiere identidad simulada mediante un mensaje *AVS*: comparación directa de dos candidatos (escenario dinámico).

ALG de los iniciadores del algoritmo y los mensajes *ALG* que redirigen los nodos bloqueados se pueden observar en la figura 5.30.a.

El mensaje (AVS, $2ta_2\varepsilon$, ta_1, $<2ta_2,5ta_5,8ta_8,1ta_1>$, F) que manda el nodo 2 es el efecto de la ejecución de la acción $firstAVS_2$ y se representa en la figura 5.31.b. Tras la recepción de ese mensaje, el nodo 1 conoce a su candidato sucesor (nodo 3) y a su candidato predecesor (nodo 2). Comparando ambos candidatos, $3ta_3\varepsilon > 2ta_2\varepsilon$, el nodo 1 debe transmitir un mensaje *AVSRSP* al nodo 2, comunicándole la superioridad del nodo 3 (ver la figura 5.32.c). La ruta del mensaje *AVSRSP* es el resultado de concatenar la ruta que parte del candidato predecesor hasta el nodo l y la que comienza en el nodo 1 y llega hasta el candidato sucesor. La estructura del mensaje (AVSRSP, ta_2, $3ta_3\varepsilon$, $<2ta_2,5ta_5,8ta_8,1ta_1,3ta_3>$) reúne toda la información que posee. Al enviarlo al nodo 2, la instancia del algoritmo que inició el nodo 1 deja de participar en la detección, o lo que es lo mismo, concluye.

Suponiendo que al mismo tiempo que el mensaje *AVSRSP* alcanza el nodo 2, se modifica el grafo de esperas del ejemplo. Si el nodo 4 se desbloquea, todo el reconocimiento de esperas y nodos del grafo, realizado por ese candidato, se podría llegar a

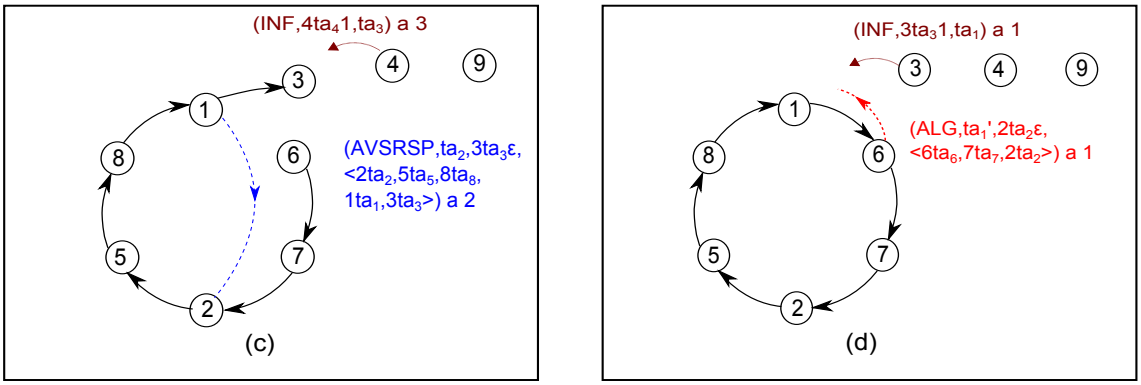

Figura 5.32: Detección por un nodo iniciador que adquiere identidad simulada mediante un mensaje *AVS*: borrado de un mensaje *INF*.

perder si no se ponen en marcha mecanismos para salvaguardar la información válida. Igual que en los ejemplos de las secciones 5.2.1 a 5.2.2.2, con la ejecución de la acción $StartDelArc_4(3,ta_3)$ se procede, mediante un mensaje *INF*, a dar aviso de que el nodo 4 interrumpe su control de la instancia del algoritmo. Se puede observar este mensaje *INF* en el grafo de la figura 5.32.c. Este mensaje *INF* nunca llegará a ser procesado porque no puede cumplir, a la vez, los requisitos temporales y de estado que precisa la acción $rcvINF_3(4,m)$. Por eso queda habilitada automáticamente la acción $dltINF_3(4,m)$ que, al igual que el ejemplo de la sección 5.2.2.2, puede ejecutarse en cualquier instante de la ejecución y encargase del borrado de mensajes *INF* no útiles.

El proceso de borrado de esperas sigue su curso tal y como se pone de manifiesto en la figura 5.32.d. Cuando se ejecuta la acción $StartDelArc_3(1,ta_1)$ se forma un nuevo mensaje *INF*. Esta vez el mensaje es de interés para el nodo 1 y para el candidato al que mandó información referente al nodo 3. La activación total del nodo 1 mediante la acción $EndDelArc_1(3)$ deja patente la necesidad de adquirir la identidad simulada $(3ta_3 1)$. Para ello, uno de los efectos de esta acción asigna el valor *unknown* a la variable $status.id_1$. Aunque la acción $rcvINF_1(3,m)$ puede retrasarse y, en consecuencia, demorar el mecanismo de adquisición de la identidad simulada, el grafo de esperas

puede seguir evolucionando. A pesar de que $status.id_1 = unknown$, el nodo 1 puede establecer una nueva relación de espera y bloquearse por el nodo 6. Al terminar de formarse esta espera (acción $EndAddArc_6(1,ta_1\prime)$), el nodo 6 envía un mensaje ALG al nodo 1 con el contenido de st_alg_6. Mientras el nodo 1 no adquiera la identidad simulada, este mensaje ALG no podrá ser procesado en el nodo destino (véase esta situación representada en la figura 5.32.d). Esto implica que el proceso de detección queda paralizado hasta que los mecanismos dinámicos concluyan.

La adquisición de la identidad simulada es un efecto de la acción $rcvINF_1(3,m)$. Antes de asumir la identidad simulada, el nodo 1 comprueba que su identidad es inferior a la que recibe. Hay que recordar que el nodo 1 antes de activarse era un nodo candidato que, además, envió un mensaje $AVSRSP$ al nodo 2. Para formar la ruta de este mensaje $AVSRSP$, el nodo 1 unió las rutas que guardaba en su st_avsrsp_1 y en su $set\_st\_avs_1$. A diferencia de la generación de un mensaje AVS, en la creación de un mensaje $AVSRSP$ no se elimina el mensaje almacenado en $set\_st\_avs_i$. Por eso, tras la activación, el nodo 1 mantiene el mensaje ($2ta_2\varepsilon$, ta_1, $<2ta_2,5ta_5,8ta_8,1ta_1>$, T) en $set\_st\_avs_1$. El último campo de este mensaje incluye una marca, T, que lo distingue de mensajes almacenados en $set\_st\_avs_1$ que no han sido usados para formar nuevos mensajes $AVSRSP$. Tras adquirir la identidad simulada, el nodo 1 transforma en mensaje $AVSRSP$ a todos aquellos mensajes almacenados en $set\_st\_avs_1$ cuyo campo $m.sid$ sea inferior a la nueva identidad del nodo 1. En este caso, como $m.sid = 2ta_2\varepsilon$ es inferior a $sim\_id_1 = 3ta_31$, se envía un mensaje $AVSRSP$ al nodo 2 con los siguientes datos: (AVSRSP, ta_2, $3ta_31$, $<2ta_2,5ta_5,8ta_8,1ta_1>$).

Esta transformación del mensaje almacenado en $set\_st\_avs_1$ en un mensaje $AVSRSP$ es una muestra del uso del mecanismo de reconversión de mensajes $AVSRSP$, implementado en el algoritmo y presentado en este ejemplo (figura 5.33.e). El objetivo fundamental de este mecanismo es reutilizar información almacenada por un nodo

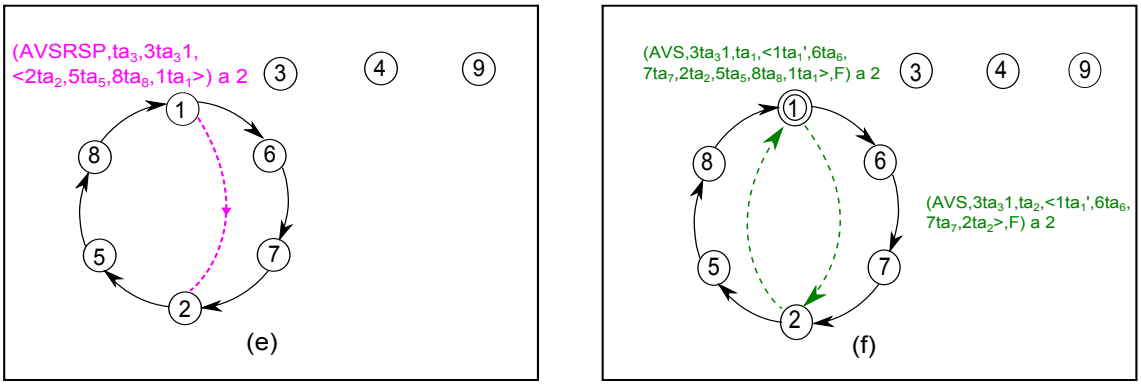

Figura 5.33: Detección por un nodo iniciador que adquiere identidad simulada mediante un mensaje *AVS*: reconversión de mensajes *AVSRSP*.

antes de que sea eliminada porque va a empezar a comportarse como un nodo distinto. Si el nodo 1 hubiera presentado la nueva identidad en la configuración inicial, al contactar con el nodo 2, se habría almacenado en st_avsrsp_2 un mensaje *ALG* con todos los nodos ubicados entre ellos. Por lo tanto, para simular este funcionamiento, el nodo 1 debe mandar un mensaje *AVSRSP* y no un mensaje *AVS*, como podría pensarse en un principio, por tratarse del reciclaje de un mensaje almacenado en $set\_st\_avs_1$. Corregir información almacenada en nodos relativa a otros que ya no forman parte de la ordenación actual y evitar que, ante una configuración en la que han cambiado algunas esperas, tenga que ejecutarse una nueva instancia del algoritmo para recoger esos cambios, son otros objetivos del mecanismo descrito.

Finalmente, la recepción de este mensaje *AVSRSP* permite seguir el proceso de detección como si de un escenario estático se tratase. La adquisión de la identidad simulada convierte al nodo 1 automáticamente en un nodo candidato. Este efecto facilita que, al recibir el mensaje *ALG* del nodo 6 que quedó pendiente, se habilite la acción $firstAVS_1$. Otra condición para que el nodo 1 pueda mandar el mensaje *AVS* de dos nodos, que se comunican por primera vez, es que la identidad simulada del nodo 1 sea superior a la identidad del candidato que generó el mensaje *ALG*. Como

$3ta_3 1 > 2ta_2 \varepsilon$, el nodo 1 dirige el mensaje (AVS, $3ta_3 1$, ta_2, $<1ta_1\prime,6ta_6,7ta_7,2ta_2>$, F) al nodo 2. En cuanto se recibe y almacena este mensaje en $set\_st\_avs_2$, el nodo 2 dispone de suficiente información para comparar a su candidato sucesor y al que le precede. La comparación de las identidades simuladas de ambos candidatos establece que se trata de un mismo nodo. Por consiguiente, el nodo 2 envía el mensaje (AVS, $3ta_3 1$, ta_1, $<1ta_1\prime,6ta_6,7ta_7,2ta_2,5ta_5,8ta_8,1ta_1>$, F) al nodo 1. Posteriormente y sin necesidad de almacenar el mensaje *AVS*, el nodo 1 detecta la existencia de un ciclo porque su identificador aparece en el primer elemento de la ruta y en el último. La fase de detección tal y como se muestra en la figura 5.33.f da paso a la fase de resolución del interbloqueo, en la que el nodo 1 tiene que abortar, esto es, romper las esperas del grafo que lo vinculan al nodo 8 y al nodo 6.

Los efectos de las acciones incluidas en la tabla 5.6 quedan representados gráficamente en la figura 5.34. Sería posible ordenar de otra forma algunas de las acciones englobadas en el epígrafe *Dinamismo* pero los mensajes necesarios para la detección del interbloqueo en ese caso no cambiarían. Por ejemplo, después de la acción $EndDelArc_3(4)$ podrían sucederse las acciones $StartDelArc_3(1,ta_1)$, $EndDelArc_1(3)$, $StartAddArc_1(6)$ y $EndAddArc_6(1,ta_1\prime)$. Este orden de ejecución permitiría que el grafo de esperas se modificara totalmente hasta formar un ciclo y después se produjeran las acciones necesarias para acomodar la información a la nueva configuración de nodo, esto es, $rcvINF_1(3,m)$ y $dltINF_3(4,m)$.

El esquema de instancias de la figura 5.35 correspondiente a este ejemplo de ejecución consiste en la evolución de las instancias lanzadas por el nodo 1, 2, 3 y 4. La instancia del nodo 1 es la primera que abandona el proceso de detección. La activación del nodo 4 precipita el cese de su instancia. Su información se pierde porque no es relevante para el curso de las instancias activas. Cuando desaparece la espera que parte del nodo 3 y se activa el nodo 1, la instancia del nodo 3 deja de estar controlada

Instancia 1-nodo 4	**Dinamismo**	**Instancia 2-nodo 3**	**Instancia 3-nodo 1**	**Instancia 4-nodo 2**
initiate_4		initiate_3	initiate_1	initiate_2
$\text{rcvALG}_3(4, m)$		$\text{rcvALG}_1(3, m)$	$\text{rcvALG}_8(1, m)$ $\text{rcvALG}_5(8, m)$ $\text{rcvALG}_2(5, m)$ $\text{rcvAVS}_1(2, m)$ sndAVSRSP_1	$\text{rcvALG}_7(2, m)$ $\text{rcvALG}_6(7, m)$ firstAVS_2 $\text{rcvAVSRSP}_2(1, m)$
	$\text{SDA}_9(4, ta_4)$ $\text{EDA}_4(9)$ $\text{SDA}_4(3, ta_3)$ $\text{EDA}_3(4)$ $\text{dltINF}_3(4, m)$ $\text{SDA}_3(1, ta_1)$ $\text{EDA}_1(3)$			
	$\text{rcvINF}_1(3, m)$		**Instancia 2(*)-nodo 1**	
	$\text{SAA}_1(6)$ $\text{EAA}_6(1, ta_1\prime)$			$\text{rcvAVSRSP}_2(1, m)$
			$\text{rcvALG}_1(6, m)$ firstAVS_1	 $\text{rcvAVS}_2(1, m)$ sndAVS_2
			$\text{rcvAVS}_1(2, m)$	
			Abort_1	

Tabla 5.6: Secuencia de ejecución para un escenario dinámico con detección de mensaje *AVS* con nueva identidad simulada

por él y pasa a ser regida por el nodo 1. Este cambio de dirección se realiza mediante la adquisición de una identidad simulada relacionada con el nodo 3. El nodo 1 que dio por terminada la instancia que lanzó se involucra de nuevo en el proceso de detección y toma el control de la instancia del nodo 3. Finalmente, el nodo 1 da por concluida la instancia que adoptó como propia y logra así la detección del interbloqueo.

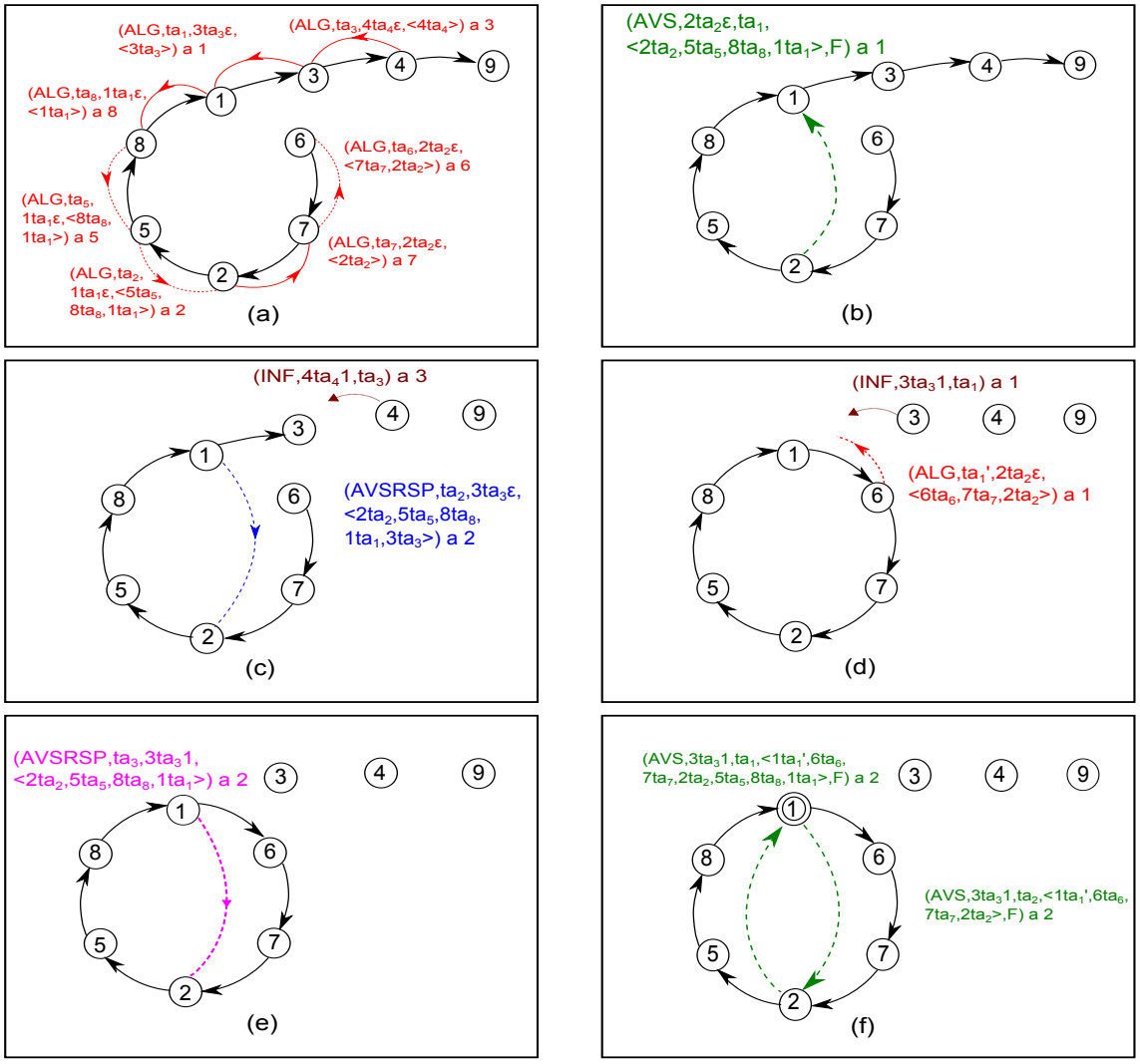

Figura 5.34: Detección por un nodo iniciador que adquiere identidad simulada mediante un mensaje *AVS*. Borrado de mensaje *INF*. Reconversión de mensajes *AVSRSP*.

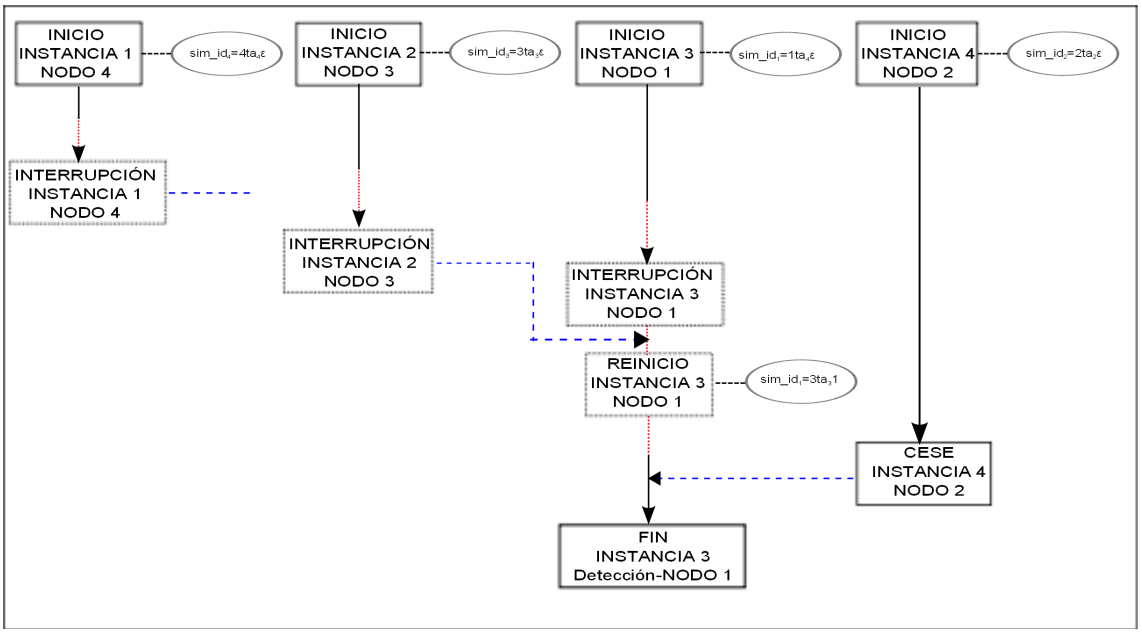

Figura 5.35: Diagrama de instancias del ejemplo de detección con *AVS* (caso dinámico).

5.2.2.4. Detección por un nodo no iniciador que adquiere una identidad simulada propagada por diversos caminos

El algoritmo que se ha diseñado es válido para el modelo de petición de único recurso. Así que, asumiendo que un nodo sólo puede bloquearse por otro nodo, se pueden plantear configuraciones en las que, cumpliéndose esta condición, haya varios nodos bloqueados por un mismo nodo. Este tipo de configuraciones tiene aspecto de árbol en el que los nodos descendientes "esperan"por un único nodo antecesor. Las ordenaciones de nodos analizadas hasta el momento son cadenas en las que, por supuesto, cada nodo tiene un sucesor y éste, a su vez, sólo cuenta con un predecesor. La ejecución del algoritmo en una configuración de nodos en árbol, que evoluciona mediante el borrado y la formación de esperas hasta formar un ciclo, va a permitir estudiar cómo se propagan y adquieren las identidades simuladas en este tipo de configuración. Este ejemplo, por tanto, resulta de interés no tanto porque utilice el

mecanismo de propagación y adquisición de identidades simuladas, sino porque este mecanismo permite generar identidades simuladas referidas a un mismo nodo para todos sus predecesores. Asimismo, cuando se comparan cualquiera dos identidades simuladas, el mecanismo debe aplicar un criterio para discernir cuál de ellas es mayor. Este criterio debe ser igualmente consistente si se comparan identidades simuladas procedentes de un mismo nodo o no, y en caso de que las identidades simuladas que se comparen correspondan a distintos instantes de propagación de una misma identidad. Resumiendo, en este ejemplo de ejecución se va a describir el **mecanismo general de generación y comparación de identidades simuladas** que tiene que estar perfectamente integrado en el mecanismo de propagación y adquisición de identidades simuladas explicado en la sección 5.1.1.

La configuración de nodos que se propone en esta ejecución se caracteriza porque el nodo 1 tiene más de un predecesor, el nodo 3 y el nodo 4 (véase la figura 5.36.a). El nodo 5, el nodo 7 y el nodo 8 de esta configuración inician el algoritmo. Mediante un mensaje ALG la identidad del nodo 8 llega al nodo 1, pero a partir del nodo 1 la propagación de la identidad del nodo 8 sigue dos caminos distintos. Como el nodo 1 tiene dos predecesores, nodo 3 y nodo 4, la identidad del nodo 8 se difundirá de manera separada por los caminos que fijan las esperas entre el nodo 1 y sus predecesores. El mensaje ALG que se transmite hacia el nodo 3 permite que la identidad del nodo 8 llegue, en primer lugar, hasta el nodo 3 y, posteriormente, a través del otro mensaje ALG, hasta el nodo 7. Por su parte, el mensaje ALG que se dirige hasta el nodo 4 contiene la identidad del nodo 8 y, seguidamente, con un nuevo mensaje ALG, la identidad del nodo 8 alcanza el nodo 5. Los nodos 5 y 7 responden de igual forma a la recepción de sus respectivos mensajes porque sus identidades son inferiores a la identidad del nodo 8. Ambos nodos almacenan en sus respectivos st_avsrsp_i las rutas que comunican a los iniciadores involucrados: $st\_avsrsp_5 = (ta_5,\ 8ta_8\varepsilon,\ <5ta_5,4ta_4,1ta_1,8ta_8>)$ y $st\_avsrsp_7 = (ta_7,\ 8ta_8\varepsilon,\ <7ta_7,3ta_3,1ta_1,8ta_8>)$.

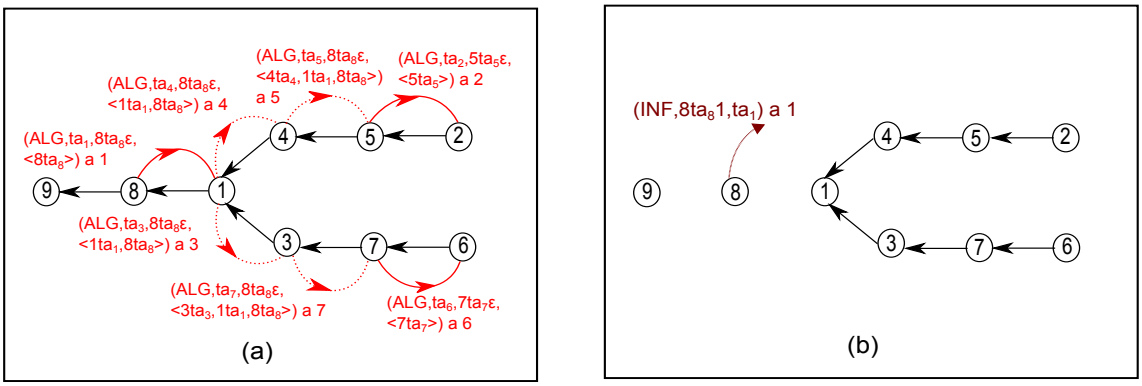

Figura 5.36: Detección por un nodo no iniciador que adquiere una identidad simulada propagada por varios caminos: (a) fase de iniciación y búsqueda de otros candidatos, (b) propagación de identidad simulada.

Antes de describir cómo varía la configuración de nodos del ejemplo, hay que mencionar también lo que sucede con las otras instancias del algoritmo. La iniciación del algoritmo por parte del nodo 5 y del nodo 7 hace posible la propagación de sus identidades. Tanto la propagación de la identidad del nodo 5 hasta el nodo 2, como la del nodo 7 hasta el nodo 6, se realiza a través de mensajes ALG. Estos mensajes ALG, tras su recepción, quedan almacenados en st_alg_2 y en st_alg_6, respectivamente. La propagación de todos los mensajes ALG siguiendo la configuración de esperas inicial se puede observar en la figura 5.36.a.

En este punto de la ejecución, el nodo 8 y el nodo 1 se activan. Para conseguir la activación de estos dos nodos tienen lugar las transiciones $StartDelArc_9(8,ta_8)$, $EndDelArc_8(9)$, $StartDelArc_8(1,ta_1)$ y $EndDelArc_1(8)$. Uno de los efectos de la acción $StartDelArc_8(1,ta_1)$ consiste en el envío de un mensaje INF al nodo 1 con la identidad simulada del nodo 8, $8ta_81$, tal y como muestra la figura 5.36.b.

Una nueva evolución del sistema supone que tanto el nodo 4 como el nodo 3 se desbloquean. Estas activaciones implican a su vez que el nodo 1 ejecuta la acción $StartDelArc_1(j,t)$ por cada uno de sus predecesores, esto es, una vez por el nodo 4 y otra por el nodo 3. Ambas ejecuciones tienen efectos similares: el envío de un mensaje

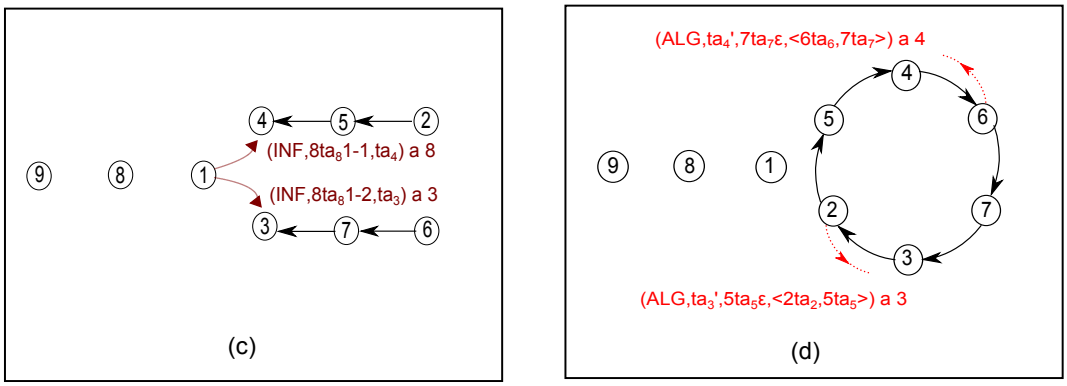

Figura 5.37: Detección por un nodo no iniciador que adquiere una identidad simulada propagada por varios caminos: (c) propagación de identidad simulada por varios caminos, (d) formación del ciclo.

INF. Sin embargo, a la hora de formar la identidad simulada que se transmite en cada uno de ellos, surge un inconveniente. En todos los ejemplos descritos hasta el momento, la identidad simulada se transmite al único predecesor que posee el emisor del mensaje *INF*, pero en este ejemplo él va a transmitir dos mensajes *INF*, uno por cada uno de sus predecesores. En ambos mensajes la identidad simulada tiene que hacer referencia al mismo nodo (nodo 8) porque el mensaje *ALG* que recibieron era consecuencia de la instancia que puso en marcha ese nodo. Se sabe que la identidad simulada permitirá, al nodo que la adopte, desempeñar el papel del nodo iniciador 8, aunque ya no esté participando en la configuración actual. Puesto que el nodo 8 difundió sus mensajes por dos caminos diferentes, hay dos nodos que tienen que asumir la identidad del mismo nodo. Como los nodos del sistema tienen que presentar una identidad única, la identidad simulada que se transmita por distintos caminos tendrá que hacer referencia al mismo nodo, pero a la vez deberá ser posible identificar la rama o el predecesor que la propagó. De acuerdo con la figura 5.37.c, la identidad simulada que viaja hasta el nodo 4 es $8ta_8$1-1 y la que se va por la rama del nodo 3 es $8ta_8$1-2. De esta forma, se logra mantener la unicidad de la identidad y compartir la identidad del nodo de la instancia que los alcanzó.

Atendiendo estos requisitos, se diseña el mecanismo de generación de la identidad simulada. Este mecanismo debe estar integrado en el de propagación de identidades simuladas. Por tanto, el proceso de generación de identidades simuladas se incluye en todas aquellas acciones del algoritmo donde se forman mensajes *INF*, esto es, $StartDelArc_i(j,t)$, $rcvINF_i(j,m)$ y $Abort_i$. La identidad simulada se compone de tres campos diferentes. En los dos primeros aparece el identificador y el tiempo de activación del nodo que se desea propagar. En este ejemplo todos los predecesores del nodo 8, ya sea directo como el nodo 1 o indirectos como el nodo 4 y el nodo 3, cuentan con $sim\_id_8.id = 8$ y $sim\_id_8.ta = ta_8$. El tercer campo de la identidad simulada,$sim\_id_i.nu$, es una cadena numérica que crece conforme se va propagando la identidad simulada por las esperas eliminadas del grafo. Inicialmente, el valor de esta componente es la cadena vacía (ε). En los escenarios estáticos las identidades simuladas de los nodos se pueden asimilar al par (identificador, tiempo de activación) porque la tercera componente de la identidad simulada siempre vale ε. En la ejecución que se está analizando, inicialmente $sim\_id_8 = 8ta_8\varepsilon$. Tras activarse y comenzar a romper la espera del bloqueo de su predecesor, el nodo 8 forma la identidad simulada $8ta_81$. Sustituyendo ε por el valor 1, se representa que la identidad se ha propagado a un predecesor del que se ha desligado. Esta identidad es la que se manda al nodo 1 para que siga participando en la configuración sin presencia del nodo 8. El nodo 1, después de admitir la identidad simulada, $sim\_id_1 = 8ta_81$, se ve inmerso en una situación parecida a la del nodo 8, cuando se activa y rompe parcialmente el bloqueo del nodo 4 por él. La identidad que manda el nodo 1 ve incrementada la longitud de su último campo en $8ta_81$-1. Además antes de proceder al envío comprueba si todavía tiene algún predecesor. Como el nodo 1 todavía está unido al nodo 3, incrementa en 1 la variable numérica $cont_1$, pasando a valer 2 (el valor inicial de $cont_1$ es 1). Este es el primer ejemplo de ejecución que hace uso de esta información.

Cuando el nodo 4 adquiere la identidad, $sim\_id_4 = 8ta_81$-1, sabe que se han roto

dos esperas entre el nodo 8 porque la longitud de $sim\_id_4.nu$ es 2. En consecuencia, el nodo 4 era, en la configuración de esperas de partida, un predecesor del nodo 8 no directo, concretamente de segundo nivel. Si se inicia el borrado de la espera del grafo que bloquea al nodo 3 por el nodo 1, la identidad simulada que genera el nodo 1 para su otro predecesor es $8ta_8$1-2. Esto implica que a la última parte de sim_id_1 se le añade el valor de $cont_1$. Si no quedan más predecesores que puedan necesitar heredar la identidad del nodo 8, la variable $cont_1$ se inicializa. En caso contrario, se incrementa en una unidad, $cont_1 = 3$. Al adquirir el nodo 3 la identidad citada, el nodo 3 asume que entre él y el nodo 8 se han roto dos esperas (longitud de la cadena $sim\_id_i.nu$ es 2) y además que, antes de activarse, era un predecesor de segundo nivel del nodo 8.

Siguiendo con la evolución del sistema, se forma un ciclo al aparecer dos nuevas esperas en la configuración (figura 5.37.d). Por un lado, el nodo 4 se bloquea por el nodo 6 ($StartAddArc_4(6)$ y $EndAddArc_6(4,ta_4\prime)$) y, por otro lado, el nodo 3 completa una espera por el nodo 2 ($StartAddArc_3(2)$ y $EndAddArc_2(3,ta_3\prime)$. En ese instante hay cuatro instancias activas para la detección del ciclo, las que iniciaron los nodos 5 y 7 y las que han adoptado los nodos 4 y 3, en representación del nodo 8 para sus dos vías de comunicación. La instancia del nodo 5 y la instancia del nodo 7 prosiguen buscando a otro candidato. Para llevar a cabo esta búsqueda los nodos 2 y 6 respectivamente difunden la identidad de estos iniciadores mediante mensajes ALG. La comparación de la identidad del nodo 7 ($7ta_7\varepsilon$) con la identidad simulada del nodo 4 ($8ta_8$1-1), hace que el nodo 4 considere al nodo 7 un candidato menor. Al igual que en los escenarios estáticos, el nodo 4 lanza un mensaje AVS al nodo 7 para indicarle que su identidad simulada es superior. Del mismo modo, el nodo 3 envía un mensaje AVS al nodo 5 ya que $sim_id_3 = 8ta_8$1-2 $> 5ta_5\varepsilon$. Los mensajes AVS que quedan representados en la figura 5.38.e son los efectos comentados de las acciones $firstAVS_4$ y $firstAVS_3$.

Cuando esos mensajes AVS llegan a su destino, se almacena ($8ta_8$1-2, ta_5, $<3ta_3\prime$,

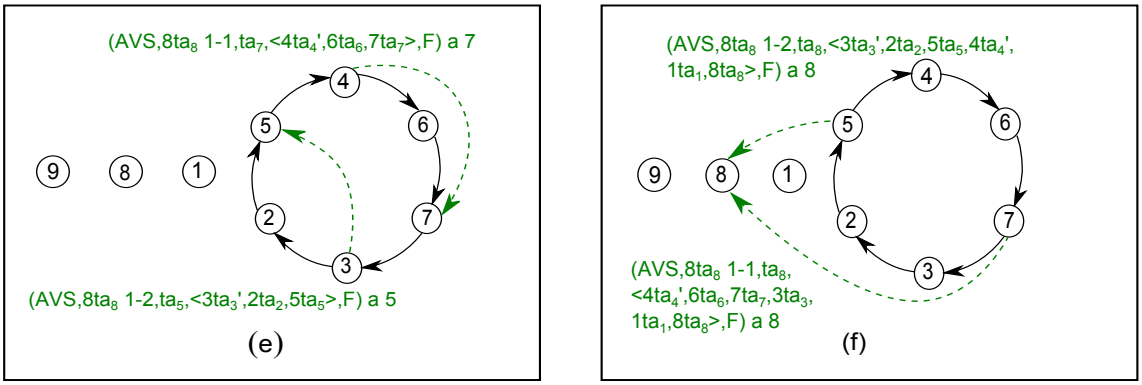

Figura 5.38: Detección por un nodo no iniciador que adquiere una identidad simulada propagada por varios caminos: (e) comparaciones directas de dos candidatos, (f) comparaciones de varios candidatos

$2ta_2$,$5ta_5$>, F) en $set\_st\_avs_5$ y ($8ta_8$1-1, ta_7, <$4ta_4\prime$,$6ta_6$,$7ta_7$>, F) en $set\_st\_avs_7$. Tanto el nodo 5 como el nodo 7 están en disposición de formar un nuevo mensaje porque cada uno cuenta con información de un candidato sucesor en sus respectivos st_avsrsp_i. Ambos nodos mandan un mensaje *AVS* al nodo 8 porque ambos lo tienen registrado como su candidato sucesor. Es obvio que, al comparar $8ta_8\varepsilon$ con $8ta_8$1-1 y con $8ta_8$1-2, se considere que éstas últimas identidades son mayores y, por tanto, se decida mandar, en ambos casos, un mensaje *AVS* al candidato sucesor. En la comparación, a igual identificador y tiempo de activación asociado, la longitud de la cadena de propagación de la identidad es determinante para decidir qué identidad es superior. Ya que la longitud del tercer campo de la identidad del candidato sucesor es nula y la cadena final de las otras dos identidades tiene 2 elementos, se establece que las identidades con cadena de longitud 2 son superiores.

Al enviar el nodo 5 el mensaje (AVS, $8ta_8$1-2, ta_8, <$3ta_3\prime$,$2ta_2$,$5ta_5$,$4ta_4$,$1ta_1$,$8ta_8$>, F) y el nodo 7 el mensaje (AVS, $8ta_8$1-1, ta_8, <$4ta_4\prime$,$6ta_6$,$7ta_7$,$3ta_3$,$1ta_1$,$8ta_8$>, F), tal y como aparece en la figura 5.38.f, los nodos 5 y 7 dan por finalizado el avance de las instancias del algoritmo que lanzaron. Una vez que los mensajes *AVS* dirigidos al nodo 8 alcanzan su destino, se pone en marcha el mecanismo de retroceso de

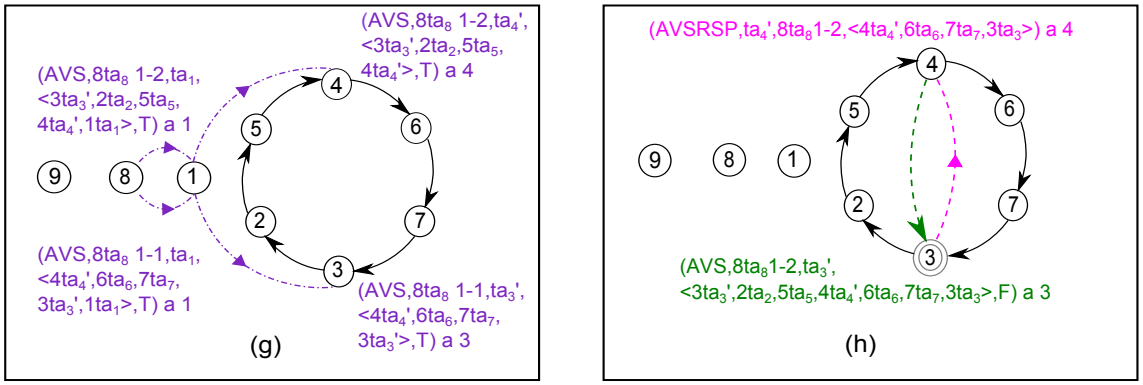

Figura 5.39: Detección por un nodo no iniciador que adquiere una identidad simulada propagada por varios caminos: (g) retroceso de mensajes *AVS*, (h) reconversión de mensaje *AVSRSP* y detección final

mensajes *AVS*. En una primera fase se forman nuevos mensajes *AVS* al nodo 1. La ruta de estos mensajes *AVS* se corrige para que no incluya la espera entre el nodos 8 y 1 que no es real, es decir, se elimina en ambos mensajes el par $8ta_8$. Recuérdese que los mensajes producidos en este mecanismo van marcados en su último campo con el valor *T*, indicando que son mensajes en retroceso. Posteriormente, la recepción de los mensajes *AVS* por parte del nodo 1 conlleva el mismo efecto. En esta ocasión, los mensajes *AVS* que se generan tienen destinos diferenciados, nodo 4 y nodo 3 respectivamente (ver figura 5.39.g).

El mensaje *AVS* que llega al nodo 4 es convenientemente almacenado en $set\_st\_avs_4$, mientras que el mensaje *AVS* que llega al nodo 3 se transforma en un mensaje *AVSRSP* al nodo 4. Esta transformación de un mensaje *AVS* en un mensaje *AVSRSP* es otro ejemplo del mecanismo de reconversión de mensajes *AVSRSP*. En la sección 5.2.2.2 se estudió este mismo mecanismo, pero incluido en la acción $rcvINF_i(j,m)$. Aunque en este ejemplo el mecanismo se invoca desde la acción $rcvAVS_i(j,m)$, la manera de convertir el mensaje *AVS* en un mensaje *AVSRSP* es idéntica a la empleada en la otra acción. El mecanismo se pone en marcha porque la identidad simulada del nodo 3 ($8ta_8$1-2) es superior a la identidad que contiene el mensaje *AVS*, $m_{AVS}.sid =$

$8ta_8$1-1. Como el candidato predecesor es menor que el nodo debe almacenar el mensaje *AVS*, le corresponde al candidato predecesor tener almacenada esta información en su st_avsrsp_i. Por eso, la información del mensaje *AVS* recibido en 3 se convierte en un mensaje *AVSRP* para el nodo 4 (figura 5.39.h). La necesidad de cambiar la ubicación de los mensajes emitidos se debe principalmente al dinamismo y a la formación y propagación de identidades simuladas.

El mensaje (AVSRSP, $ta_4\prime$, $8ta_8$1-2, $<4ta_4\prime,6ta_6,7ta_7,3ta_3>$), que contiene información actualizada del candidato sucesor del nodo 4, queda escrito en st_avsrsp_4. Combinando adecuadamente la información de que dispone el nodo 4 en $set\_st\_avs_4$ y en st_avsrsp_4, se forma un mensaje *AVS* al nodo 3 con el que se da por extinguida la instancia que asumió el nodo 4. La ruta ($<3ta_3\prime,2ta_2,5ta_5,4ta_4\prime,6ta_6,7ta_7,3ta_3>$) del mensaje *AVS* generado por 4, representado en la figura 5.39.h, contiene un ciclo. Cuando el nodo 3 recibe el mensaje en cuestión se produce la detección y será el mismo nodo 3 el que resuelva el interbloqueo abortando su participación en el sistema.

Todas las acciones que se ejecutan en este ejemplo aparecen en la tabla 5.7 organizadas según la instancia en la que se desarrollan. Junto con esta tabla se muestra la figura 5.40 con la representación completa de todos los mensajes que surgen en la ejecución de esas acciones. En la misma figura se puede observar la evolución de las esperas del grafo que acaba, finalmente, en la formación del ciclo a detectar.

En el esquema de la figura 5.41, se muestran las instancias que intervienen en la detección de este ejemplo. Inicialmente, los nodos 5, 7 y 8 lanzan instancias del algoritmo. Mientras las dos primeras siguen su curso con normalidad, la instancia del nodo 8 se ve afectada por los cambios que experimenta el grafo de esperas. Como el nodo 8, precisamente, se desvincula de la configuración de nodos, se ponen en marcha los mecanismos para que traspase su información a otros nodos que sigan perteneciendo a la configuración. En este caso la instancia del nodo 8 debe ceder el

Inst. 1-nodo 5	**Dinamismo**	**Inst. 2-nodo 8**			**Inst. 3-nodo 7**
initiate_5		initiate_8			initiate_7
$\text{rcvALG}_2(5, m)$		$\text{rcvALG}_1(8, m)$ $\text{rcvALG}_4(1, m)$ $\text{rcvALG}_3(1, m)$ $\text{rcvALG}_5(4, m)$ $\text{rcvALG}_7(3, m)$			$\text{rcvALG}_6(7, m)$
	$\text{SDA}_9(8, ta_8)$ $\text{EDA}_8(9)$ $\text{SDA}_8(1, ta_1)$ $\text{EDA}_1(8)$ $\text{SDA}_1(4, ta_4)$ $\text{SDA}_1(3, ta_3)$ $\text{rcvINF}_1(8, m)$ $\text{EDA}_4(1)$ $\text{EDA}_3(1)$				
	$\text{rcvINF}_4(1, m)$		**Inst. 2a(*)-nodo 4**		
	$\text{rcvINF}_3(1, m)$			**Inst. 2b(*)nodo 3**	
	$\text{SAA}_3(2)$ $\text{EAA}_2(3, ta_3\prime)$ $\text{SAA}_4(6)$ $\text{EAA}_6(4, ta_4\prime)$				
$\text{rcvALG}_3(2, m)$ $\text{rcvAVS}_5(3, m)$ sndAVS_5			firstAVS_4	firstAVS_3	$\text{rcvALG}_4(6, m)$ $\text{rcvAVS}_7(4, m)$ sndAVS_7
	$\text{rcvAVS}_8(5, m)$ $\text{rcvAVS}_8(7, m)$ $\text{rcvAVS}_1(8, m)$ $\text{rcvAVS}_1(8, m)$				
			$\text{rcvAVS}_4(1, m)$ $\text{rcvAVSRSP}_4(3, m)$ sndAVS_4	$\text{rcvAVS}_3(1, m)$	
				$\text{rcvAVS}_3(4, m)$	
				Abort_3	

Tabla 5.7: Secuencia de ejecución para un escenario dinámico con detección de mensaje *AVS* con identidad simulada propagada por varios caminos.

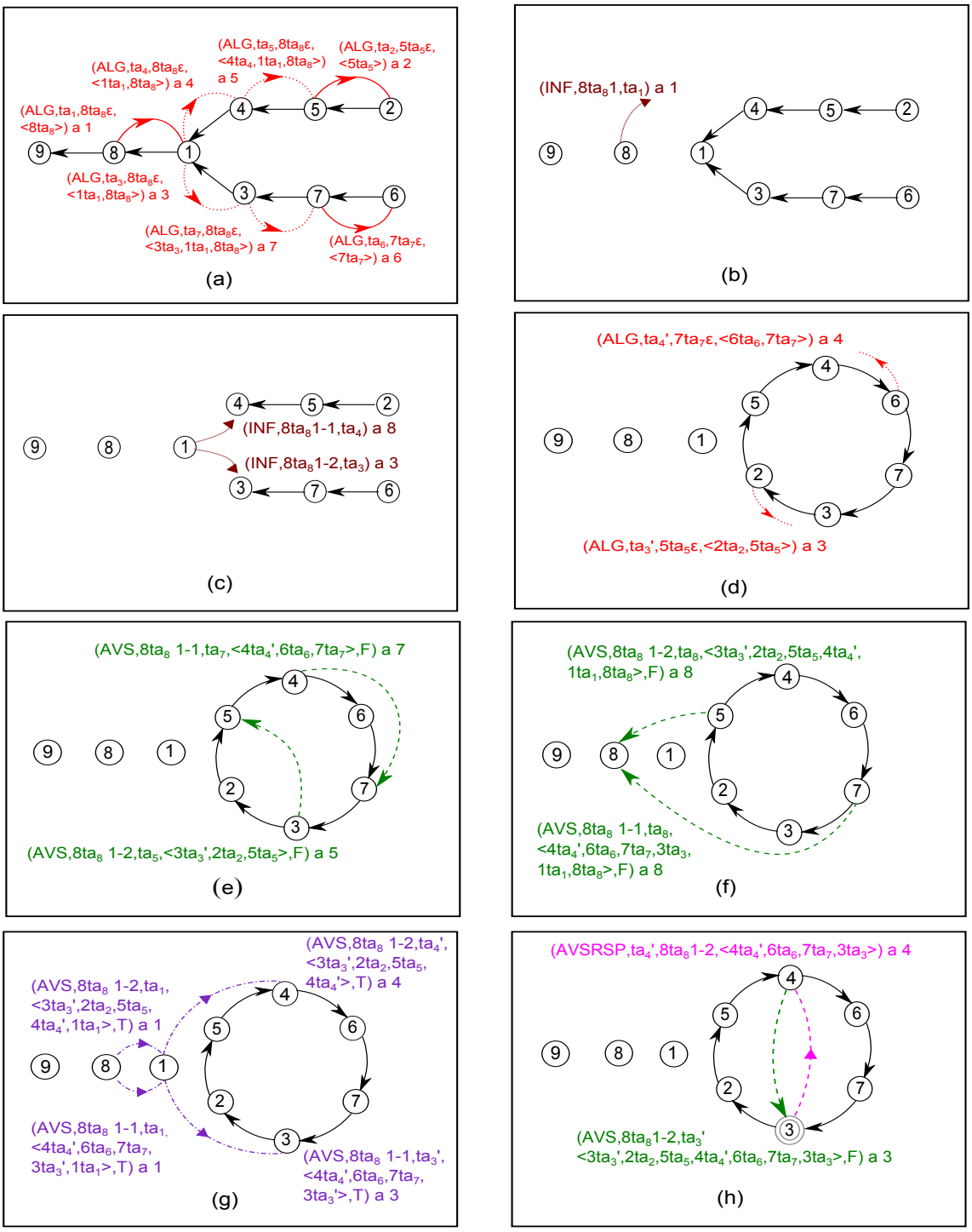

Figura 5.40: Detección por un nodo no iniciador que adquiere una identidad simulada propagada por varios caminos.

control a dos nodos diferentes, nodo 3 y nodo 4. En ejemplos anteriores el nodo que deja de participar en la configuración transfiere el control de la instancia a un sólo nodo, simulando que la instancia sigue desarrollándose sin interrupción. En cambio, en este ejemplo, la instancia del nodo 8 debe desdoblarse para representar las dos líneas de propagación que siguieron sus mensajes. En consecuencia, cuando el nodo 8 deja de dirigir la instancia que inició, ésta se divide en dos subinstancias. Cada subinstancia simula, de manera independiente, que la instancia del nodo 8 continúa ejecutándose. La evolución del grafo de esperas llega a su fin cuando se forma un ciclo. Entonces la instancia del nodo 7 da por finalizada su ejecución y traspasa la información que acumuló hasta ese momento al nodo 3.

Similarmente, el nodo 5 entrega su información al nodo 4 cuando detiene la ejecución de su instancia. A partir de ese instante sólo quedan activas las dos subinstancias derivadas de la instancia del nodo 8. Aunque podría pensarse que, de las dos subinstancias, la que tiene más posibilidades de llevar a cabo la detección del interbloqueo es la que desarrolla el nodo 4, hay que descartarla porque el nodo 3 cuenta con la identidad simulada mayor. Por lo tanto, cuando las dos subinstancias comparan sus identidades simuladas, la que corresponde al nodo 4 debe parar su ejecución en beneficio de la del nodo 3. Finalmente, la instancia que simula a la del nodo 8 y es ejecutada por el nodo, concluye el proceso de detección. El hecho de que la identidad simulada del nodo 3 sea superior a la del nodo 4 tiene que ver con la cadena de propagación de la identidad del nodo 8. En ambos casos la identidad simulada ha superado dos esperas rotas en su propagación respectiva hasta el nodo 3 y el nodo 4, pero el nodo 3 dejó de bloquear a su predecesor en un instante posterior al que lo hizo el nodo 4. En resumen, la relación de orden establecida para identidades simuladas con cadena de propagación de igual longitud viene determinada por el orden en el que se comienza a romper la espera que los une con cada uno de sus predecesores.

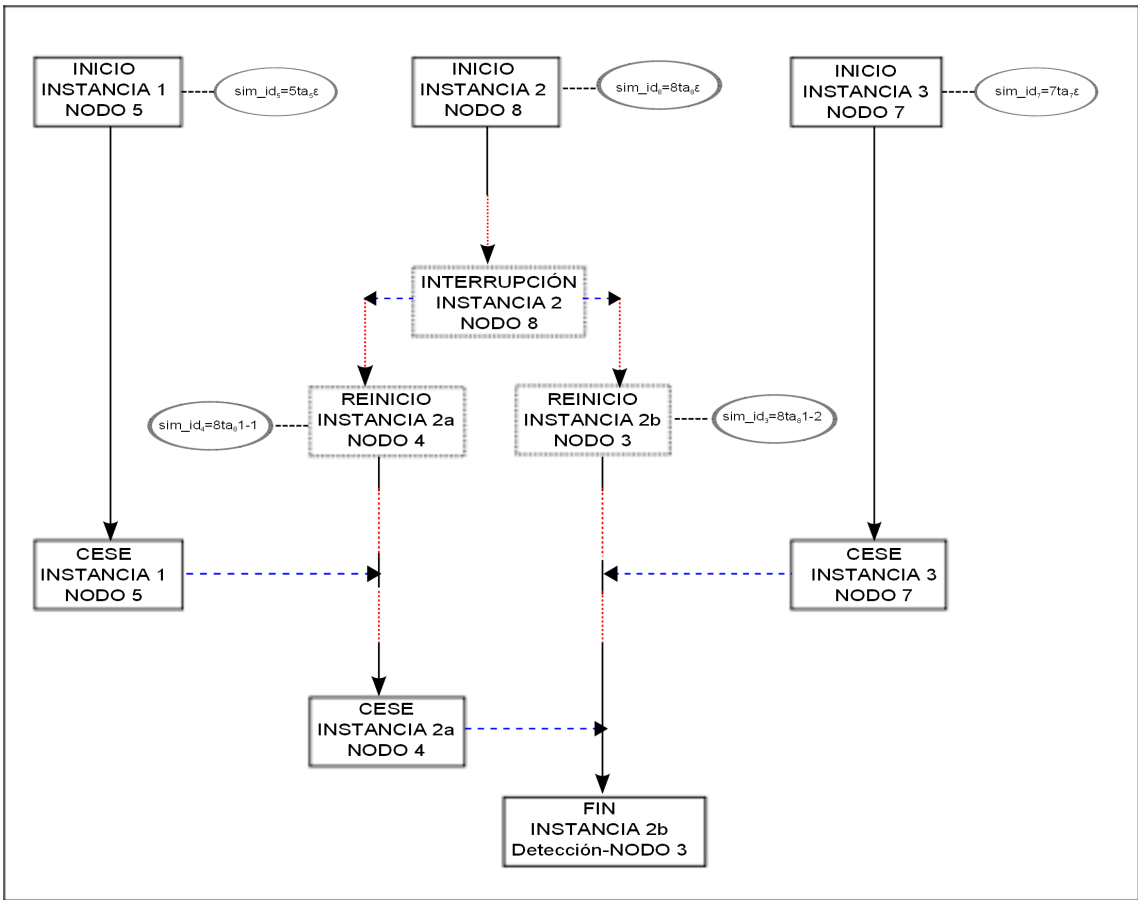

Figura 5.41: Diagrama de instancias del ejemplo de detección con *AVS* (caso dinámico).

5.2.2.5. Detección de un ciclo tras la ruptura de un ciclo previo

Como último ejemplo de ejecución en un escenario dinámico, se muestra la detección y resolución de un ciclo que se ha generado en el sistema después de haberse detectado y resuelto un interbloqueo previo. Este caso se plantea para que pueda apreciarse debidamente otra de las ventajas del carácter dinámico del algoritmo que aquí se presenta. Dado que el sistema evoluciona, una cadena de nodos resultante de la ruptura de un ciclo puede reorganizarse de nuevo y conformar otro ciclo. Obviamente, en el nuevo ciclo hay cabida para la incorporación de cualquier nodo del

sistema, a excepción del nodo abortado, y también es posible que se desligue algún nodo de los que quedaron de la resolución del primer ciclo.

Para conseguir que el algoritmo resuelva correctamente un nuevo interbloqueo que incorpora parte del camino de un interbloqueo anterior y que lo haga de una manera eficiente, es necesario que los mecanismos dinámicos, ya explicados en los ejemplos anteriores, entren en funcionamiento. Además, considerando que la configuración de partida es un ciclo roto, el éxito y la rapidez en la detección del nuevo ciclo recaerá en gran medida en los efectos de la acción $Abort_i$. Tanto el **mecanismo de propagación de identidades simuladas** como el de **retroceso de mensajes *AVS*** que incluye la acción $Abort_i$ permiten reutilizar la información recopilada por el nodo abortado antes de que éste quede aislado del sistema. El tratamiento de esa información es vital para que el algoritmo se adapte a la nueva configuración. Con el fin de que una instancia del algoritmo continúe su ejecución y llegue a detectar el nuevo interbloqueo, se aprovechan los mensajes que fluyeron por el sistema. En este punto es donde se deja entrever una de las ventajas de este algoritmo frente a otras soluciones [112].

La configuración inicial de este ejemplo es la que se aprecia en la figura 5.42.a. En ella se pueden observar gráficamente los efectos del proceso de resolución del interbloqueo mostrado en el la figura 5.40. Recuérdese que el caso anterior concluía con la detección por parte del nodo 3 del ciclo formado por los nodos 3, 2, 5, 4, 6 y 7. En ese interbloqueo el nodo 3 detectaba la situación y se proclamaba víctima. La acción $Abort_3$ es la que hace posible que el ciclo se rompa y el nodo 3 pase finalmente a estado *aborted*. La ruptura del ciclo tiene lugar gracias a que la espera que une al nodo 3 con su nodo predecesor, esto es, el nodo 7 desaparece parcialmente. Al mismo tiempo, la espera con el nodo sucesor (nodo 2) queda eliminada en su tramo inicial. De todos los efectos que se incluyen en la resolución del interbloqueo hay que destacar los que consisten en salvaguardar la información que se posee en ese instante. Esta tarea se lleva a cabo poniendo en marcha los mecanismos dinámicos de retroceso de

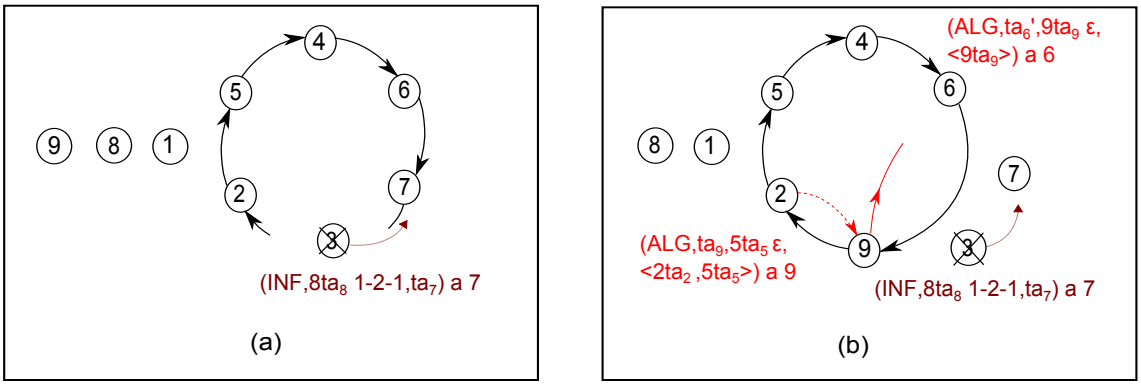

Figura 5.42: Detección de un nuevo ciclo tras la ruptura de un ciclo previo: (a) resolución del interbloqueo previo, (b) formación del nuevo ciclo.

mensajes *AVS* que el nodo 3 tenga almacenados en $set\_st\_avs_3$ y la propagación de la identidad simulada del nodo 3, mediante mensajes *INF* a todos aquellos nodos contenidos en $setPredToInf_3$. Como en $set\_st\_avs_3$ no hay almacenado ningún mensaje de tipo *AVS*, sólo se representa en la figura 5.42.a el mensaje *INF* dirigido al nodo 7. La identidad simulada que se transmite es exactamente $8ta_8$1-2-1. Parte de esa identidad ($8ta_8$1-2) fue adquirida por el nodo 3 como resultado de la evolución dinámica que finalizó al formarse el ciclo detectado. La cadena de propagación de la identidad tiene un elemento más que representa la espera rota entre el nodo 7 y el nodo 3.

El resto de efectos de la acción $Abort_3$ están relacionados con la inicialización de las variables asociadas al nodo 3: $setPred_3 = \emptyset$, $setPredToInf_3 = \emptyset$, $sim\_id_3 = 3ta_3\prime\prime\varepsilon$, $cont_3 = 1$, $st\_avsrsp_3 = NULL$ y $nofirstAVS_3 = true$.

La espera que partía del nodo 3 y la espera que finalizaba en el nodo 3 desaparecen totalmente tras la ejecución de las acciones $StartDelArc_2(3,ta_3\prime)$ y $EndDelArc_7(3)$ respectivamente. La primera acción hace que el nodo 3 desaparezca tanto del conjunto de predecesores del nodo 2, $setPred_2$, como del conjunto de predecesores con los que éste contactó, $setPredToInf_2$. En cambio, los efectos de la acción $EndDelArc_7(3)$ cambian el tiempo de activación del nodo 7 y modifican su estado a *active-unknown*. Con este

nuevo estado, el nodo 7 queda pendiente de recibir una nueva identidad simulada para seguir participando en la configuración de nodos. La identidad simulada que espera es la que el nodo 3, antes de abandonar el sistema, le envía mediante un mensaje *INF* ya que el nodo 3 le había pasado información con anterioridad. La ejecución de estas dos acciones completa los efectos la acción $Abort_3$.

La evolución dinámica que se propone después del aborto del nodo 3 queda representada en la figura 5.42.b. Esta representación evidencia que el nodo 9 establece una relación de espera con el nodo 2. Por otra parte, el nodo 6 se activa, desligándose adecuadamente del nodo 7 y seguidamente se bloquea también por el nodo 9. Todas las modificaciones del grafo de esperas que provocan esta serie de bloqueos y activaciones conducen a la formación de un nuevo ciclo en el sistema, que el algoritmo tiene que ser capaz de detectar y resolver. El bloqueo del nodo 9 por el nodo 2 tiene lugar tras la ejecución secuencial de las acciones $StartAddArc_9(2)$ y $EndAddArc_2(9,ta_9)$. Como efecto directo de la creación de esta nueva espera, el nodo 2 genera un mensaje ALG con destino el nodo 9 para traspasarle la información sobre el candidato 5 que guardó en st_alg_2. El contenido de ese mensaje ALG, tal y como se observa en la figura 5.42.b, incluye la ruta $<2ta_2,5ta_5>$ y la identidad simulada del nodo candidato que inició el algoritmo, esto es, $5ta_5\varepsilon$.

Por otra parte, la desactivación del nodo 6 se logra por medio de los efectos de las transiciones $StartDelArc_7(6,ta_6)$ y $EndDelArc_6(7)$ en ese orden. La acción $StartDelArc_7(6,ta_6)$ provoca la desaparición del nodo 6 de $setPred_7$ pero no de $setPredToInf_7$ porque está aún pendiente de recibir el mensaje *INF* del nodo 3. La acción $EndDelArc_6(7)$, que se ejecuta seguidamente, cambia el tiempo de activación del nodo 6 a $ta_6\prime$ y modifica su estado a *active-unknown*. En ese estado el nodo 6 puede bloquearse de nuevo, es decir, la configuración de nodos en la que está incluido puede seguir modificándose. A pesar de ello, el intercambio de información con otros nodos queda suspendido hasta que el valor *unknown* de su variable *status.id* cambie

a *known*. Sólo tras la recepción de un mensaje *INF*, $status.id_6$ pasa a ser *known*. La identidad simulada que debe asumir el nodo 6, después del aborto del nodo 3, es transportada en ese mensaje *INF*. Tal y como se ha mencionado anteriormente, la formación del nuevo ciclo tiene lugar cuando el nodo 6 se bloquea por el nodo 9. La ejecución consecutiva de las acciones $StartAddArc_6(9)$ y $EndAddArc_9(6,ta_6\prime)$ crea el nuevo interbloqueo. Los efectos conjuntos de estas acciones hacen que $status.alg_6 = blocked$ manteniendo $status.id_6$ a *unknown* y que el par $(6,ta_6\prime)$ se añada al conjunto de predecesores del nodo 9.

Es precisamente en ese punto de ejecución cuando se fuerza al nodo 9 a lanzar una nueva instancia del algoritmo. La acción $initiate_9$ convierte al nodo 9 en un candidato a detector del nuevo interbloqueo. Para ello manda al nodo 6 un mensaje ALG con el que pretende difundir su identidad y averiguar si existe otro candidato en el ciclo con más posibilidades de detectar y resolver el interbloqueo. El tratamiento del mensaje ALG, que se generó al formar la espera del nodo 9 al nodo 2, se retarda a propósito hasta que se hayan completado los efectos de la acción $initiate_9$. El objetivo de retrasar la acción $rcvALG_9(2,m)$ es permitir que un nodo que no participó en el primer interbloqueo resuelto se incorpore como candidato a detector en el ciclo recién formado.

En la figura 5.43.c, se representa el instante en el que el mensaje *INF* que viaja al nodo 7 todavía no ha sido retirado del canal y el mensaje ALG al nodo 6 tampoco ha sido procesado. Además, en esta misma figura el nodo 9 manda un mensaje con el siguiente contenido (AVS, $9ta_9\varepsilon$, ta_5, $<9ta_9,2ta_2,5ta_5>$, F). Este mensaje AVS que va del nodo 9 al 5 es el resultado de ejecutar la acción $firstAVS_9$ y en sus extremos se localizan los dos nodos candidatos que se han puesto en contacto. Esta acción queda habilitada tras el procesado del mensaje ALG mediante la acción $rcvALG_9(2,m)$. Como $status.alg_9 = candidate$ en el momento de recibir el mensaje ALG, éste ya no será redirigido al predecesor del nodo 9, se procederá a comparar a los dos candidatos

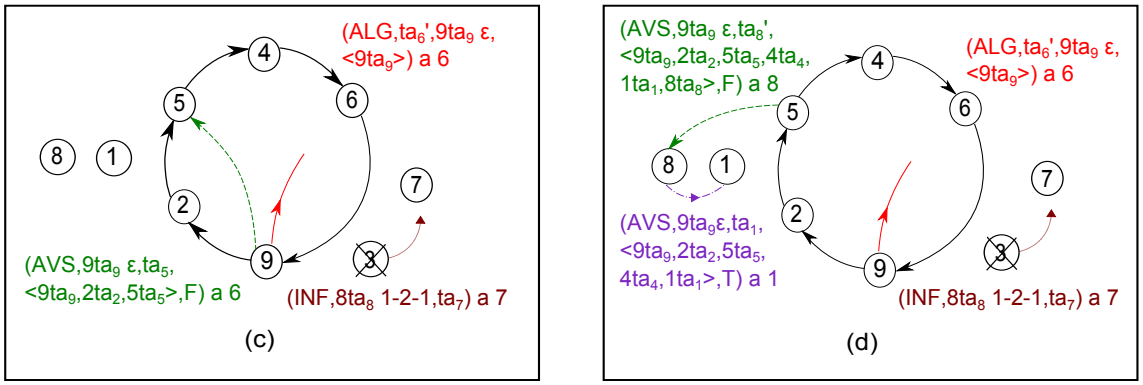

Figura 5.43: Detección de un nuevo ciclo tras la ruptura de un ciclo previo: (c) comparación directa de dos candidatos, (d) retroceso de mensaje *AVS*.

que han entrado en contacto a través de este mensaje. Por un lado, el candidato emisor del mensaje *ALG*, el nodo 5, se anotará como candidato sucesor del nodo 9, $cand\_succ_9 = 5ta_5\varepsilon$. Al comprobar que la identidad del candidato sucesor es inferior a su propia identidad simulada ($9ta_9\varepsilon$), el mensaje *ALG* se almacena en st_alg_9 tal cual ha sido recibido, $st\_alg_9 = (5ta_5\varepsilon, <2ta_2,5ta_5>)$. Verificada la superioridad del nodo candidato 9 frente al nodo 5, el nodo 9 enviará notificación al nodo 5 para que este último deje de considerarse candidato a detector. Esta notificación se produce en la transición $firstAVS_9$ y queda marcada en la variable *booleana* $nofirstAVS_9 = false$.

La recepción por parte del nodo 5 del aviso del nodo 9 desencadena la transmisión de una serie de mensajes de tipo *AVS*, tal y como se observa en la figura 5.43.d. En primer lugar, el mensaje *AVS* se almacena en $set\_st\_avs_5$ y automáticamente se ejecuta la acción $sndAVS_5$ porque el nodo 5 guardaba información en su st_avsrsp_5 de un candidato mayor con el que contactó. Dado que el nodo 5 posee información de dos candidatos a detector a parte de él, el nodo 5 puede establecer cuál de los otros dos candidatos es el de mayor identidad simulada. Si, como en este caso, la información de st_avsrsp_5 hace referencia al candidato sucesor del nodo 5, $cand\_succ_5 = 8ta_8\varepsilon$, que es claramente inferior al nodo candidato 9, $9ta_9\varepsilon$, que aparece en el mensaje almacenado

en $set\_st\_avs_5$, los almacenes se combinan para formar un mensaje de tipo *AVS* que va del nodo 5 al nodo 8. Este mensaje (AVS, $9ta_9\varepsilon$, ta_8, $<9ta_9,2ta_2,5ta_5,4ta_4,1ta_1,8ta_8>$, F) pone de manifiesto que el nodo 9 es el único nodo de los tres implicados con posibilidades de llegar a ser el detector del interbloqueo. Obviamente, si la relación de los candidatos conocidos por el nodo 5 fuera al revés, el mensaje enviado sería de tipo *AVSRSP* e iría dirigido al nodo candidato que indica el mensaje de $set\_st\_avs_5$. Cuando el nodo 8 recibe el mensaje *AVS* del nodo 5, no lo almacena en $set\_st\_avs_8$ porque se percata de que la información que recoge en su ruta no se corresponde con su identidad actual. Por tanto, queda comprobado que el nodo 1, con el tiempo de activación que aparece en la ruta del mensaje *AVS*, $1ta_1$, no pertenece al conjunto de predecesores del nodo 8. En consecuencia, se pone en marcha el mecanismo para hacer retroceder a un mensaje *AVS* siguiendo su ruta en sentido inverso. Por ello, el mensaje *AVS* que representa este retroceso en la figura 5.43.d tiene como destino el penúltimo nodo de la ruta, el nodo 1. El campo *fwd* del mensaje va marcado con *T* (*true*) y la ruta del mensaje excluye al nodo 8. En esta misma figura se muestra cómo tanto el mensaje *INF* que va del nodo 3 al nodo 7 como el mensaje *AlG* dirigido al nodo 6 procedente del nodo 9, no han llegado todavía a sus destinos.

En la figura 5.44.e se plasman los efectos de la recepción por el nodo 1 del mensaje *AVS* en retroceso. De la misma forma que sucedía con el mensaje *AVS* que llegó al nodo 8, el mensaje *AVS* que alcanza el nodo 1 no se puede almacenar en $set\_st\_avs_1$ porque la información de su ruta indica que el nodo 4, junto con su tiempo de activación, ta_4, forman parte del conjunto de predecesores del nodo 1 y, en el instante que se analiza, eso ha dejado de ser cierto. De nuevo, el mecanismo de retroceso de un mensaje *AVS* se pone en funcionamiento y el nodo 1 manda un mensaje *AVS* precisamente al nodo 4, con la señal de *AVS* en retroceso, $fwd = T$, y sin el par (nodo, tiempo) relativo al nodo 1 en la ruta del mensaje. Una vez que este mensaje *AVS* llega al nodo 4, se almacena en $set\_st\_avs_4$ porque la secuencia de pares de la ruta se

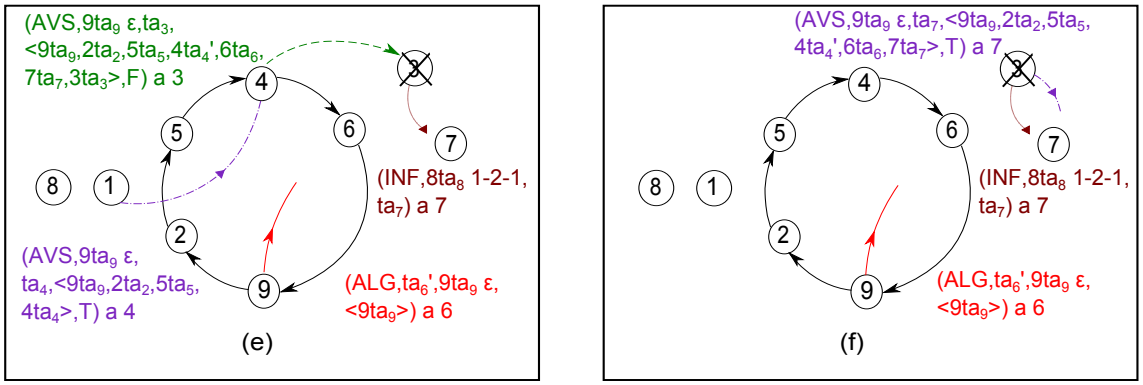

Figura 5.44: Detección de un nuevo ciclo tras la ruptura de un ciclo previo: (e) y (f) Retroceso de mensajes *AVS*.

corresponde con esperas reales del grafo. Tras la ejecución de $rcvAVS_4(1,m)$, el nodo 4 está en condiciones de comparar los candidatos a detector de los que tiene conocimiento. En $set\_st\_avs_4$ hay información del nodo candidato 9, $(9ta_9\varepsilon)$, y, por otro lado, la identidad del candidato sucesor del nodo 4 es $(8ta_8 1\text{-}2)$. Según estos datos, el nodo 4 combina la ruta del mensaje almacenado en $set\_st\_avs_4$ con la ruta de st_avsrsp_4 para formar el mensaje (AVS, $9ta_9\varepsilon$, ta_3, $<9ta_9,2ta_2,5ta_5,4ta_4\prime,6ta_6,7ta_7,3ta_3>$, F) que lanza hasta el nodo 3. Como es bien sabido, el nodo 3 es el nodo cuyo aborto resolvió el interbloqueo mostrado en la sección 5.2.2.4 y que se retomó como caso inicial para este ejemplo. Los otros dos mensajes que se observan en la 5.44.e son el mensaje *INF* al nodo 7 y el mensaje *ALG* al nodo 6, que todavía no han sido retirados de los canales correspondientes.

A continuación se puede ver en la figura 5.44.f cómo el nodo 3 no admite el mensaje *AVS* procedente del nodo 3 y lo hace retroceder hasta el nodo 7. Aunque el nodo 3 dejó de participar activamente en el sistema cuando abortó, el nodo 3 puede hacer que ese mensaje sea traspasado al penúltimo nodo de la ruta e intentar salvar la mayor parte de la información que contiene. Este proceso es una muestra más del retroceso de mensajes *AVS*, ya explicado, pero el nodo que lo desencadena, a diferencia de casos anteriores, es un nodo abortado. Tanto el mensaje *INF* que se dirige al nodo 7 como

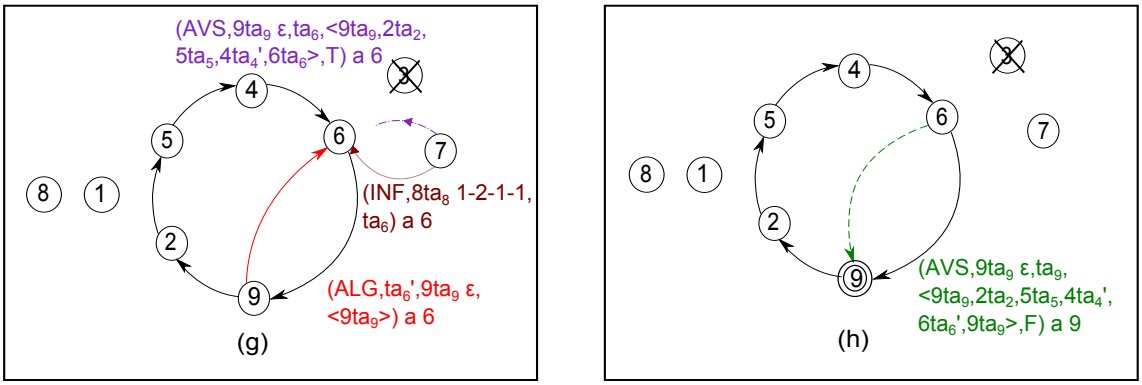

Figura 5.45: Detección de un nuevo ciclo tras la ruptura de un ciclo previo: (g) adquisición de identidad simulada, (h) detección final.

el mensaje *ALG* que va al nodo 6, que se observan también en la figura 5.44.f, son mensajes que aún no han alcanzando su destino.

En cuanto el mensaje *INF* del nodo 3 alcanza el nodo 7, los efectos de la acción $rcvINF_7(3,m)$ modifican inicialmente la identidad simulada del nodo 7, $8ta_8$1-2-1, cambian la característica de estado $status.id_7$ a *known* y originan un mensaje *INF* al nodo 6, representado en la figura 5.45.g, con una identidad simulada creada a partir de la adquirida por el nodo 7. En la anterior configuración el nodo 7 era un nodo candidato que había emitido un mensaje *AVS* al comparar dos candidatos a detector de los que tenía conocimiento. La señal de ello es que la variable inf_need_7 mantiene su valor a *true*. Antes de asumir la identidad que viaja en el mensaje *INF*, se comprueba que ésta es superior a la identidad simulada del nodo 7 anterior a la recepción del mensaje *INF*. Concretamente, se compara $m_{INF}.sid$ de valor $8ta_8$1-2-1 con $sim\_id_7 = 7ta_7\varepsilon$. A la vista de las identidades comparadas, la identidad simulada que transporta el mensaje *INF* será la nueva identidad simulada del nodo 7. En cuanto al envío de nuevos mensajes *INF* a nodos que sólo estén contenidos en el conjunto de predecesores a los que informar, el nodo 7 está obligado a mandar un mensaje *INF* al nodo 6 que es el único nodo que ya no pertenece a $setPred_7$ pero sí a $setPredToInf_7$.

La identidad simulada que se transmite al nodo 6 es $8ta_8$1-2-1-1. Se puede apreciar que esta identidad simulada coincide con la que le llega al nodo 7, salvo el último campo que ve incrementada su longitud. La cadena de números del campo final de la identidad simulada tiene que ver con el número de nodos (longitud de la cadena) que reconocieron como mayor candidato al nodo 8, mientras su tiempo de activación era ta_8, un total de cuatro nodos en este caso.

En la misma figura se puede observar la existencia de un mensaje *AVS* en retroceso, que va desde el nodo 7 al nodo 6. Este mensaje surge en el momento de la recepción del mensaje *AVS* que mandó el nodo 3 al nodo 7, $rcvAVS_7(3,m)$. En la ruta de este mensaje, $<9ta_9,2ta_2,5ta_5,4ta_4\prime,6ta_6,7ta_7>$, está incluida una espera entre el nodo 6 y el nodo 7, que ya no está en vigor, por lo que el mensaje recoge información falsa. La presencia de información incorrecta en la ruta del mensaje impide que el nodo 7 lo almacene en $set\_st\_avs_7$ y, en lugar de este efecto, se lanza un nuevo mensaje *AVS* en retroceso con destino el nodo 6 eliminando de la ruta la espera no válida, $<9ta_9,2ta_2,5ta_5,4ta_4\prime,6ta_6>$. Finalmente, hay que recordar que ni el mensaje *ALG* ni el mensaje *AVS* serán tratados por el nodo 6 hasta que la acción $rcvINF_6(7,m)$ sea ejecutada. Como el nodo 6 quedó pendiente de recoger una nueva identidad simulada, su estado $status.id_6$ será *unknown* hasta recibir el mensaje *INF* del nodo 7. En ese estado, la recepción de mensajes tipo *ALG*, *AVS* y *AVSRSP*, que sirven para intercambiar información de los candidatos a detector, queda suspendida porque las comparaciones entre candidatos se hacen a partir de sus identidades simuladas actualizadas convenientemente.

Una vez que el nodo 6 retira del canal el mensaje *INF*, éste adquiere la identidad simulada $8ta_8$1-2-1-1. Con esta nueva identidad el nodo 6 seguirá participando en el proceso de detección del ciclo del sistema, ya que su estado $status.id_6$ pasa a ser *known*. La variable $status.alg_6$ también se modifica como efecto de $rcvINF_6(7,m)$, pasando a ser $status.alg_6 = candidate$. Este cambio de estado del nodo 6 permite incluirlo en el

grupo de candidatos de entre los que saldrá elegido el detector del ciclo. El paso a candidato del nodo 6 es automático porque la formación de la espera que representa el bloqueo del nodo 6 por el nodo 9 se ha completado antes de que todos los nodos del ciclo conozcan sus identidades simuladas, es decir, antes de que *status.id* sea *known* para todos los nodos del ciclo. Este hecho posibilita que la evolución dinámica del sistema que tiene que ver con la aparición y la desaparición de esperas no se frene. Sin embargo, el intercambio de información para el proceso de detección no se simultanea con la construcción del grafo de esperas y requiere de ciertas condiciones para producirse.

Siendo $status.id_6 = known$, el mensaje AVS originado por el nodo 7 se almacena en $set\_st\_avs_6$ como efecto de la acción $rcvAVS_6(7,m)$. Por otra parte, también se ejecutan los efectos de la acción $rcvALG_6(9,m)$ ya que $status.alg_6 = candidate$. En primer lugar, se extrae información del mensaje ALG y se anota como candidato sucesor del nodo 6 al nodo 9, más concretamente, $cand\_succ_6 = 9ta_9\varepsilon$. Posteriormente, se comprueba si la identidad simulada del nodo 6 ($8ta_8$1-2-1-1) es superior a la del candidato sucesor que acaba de conocerse gracias al mensaje ALG. En este caso, como el candidato sucesor es obviamente mayor que la identidad simulada del nodo 6, se guarda en st_avsrsp_6 el mensaje ALG modificando su ruta para que aparezca en ella el nodo 6, st_avsrsp_6 = ($ta_6\prime$, $9ta_9\varepsilon$, <$6ta_6\prime$,$9ta_9$>).

Considerando la información que posee el nodo 6 tanto en $set\_st\_avs_6$ como en st_avsrsp_6, después del intercambio de mensajes realizado, es evidente que éste está en condiciones de enviar un mensaje al nodo 9. El mensaje que se mandará al nodo 9 es de tipo AVS porque la identidad simulada del mayor candidato del mensaje de $set\_st\_avs_6$, $m.sid$, coincide con la identidad de $cand\_succ_6$. La ruta de este mensaje se forma concatenando la ruta de $set\_st\_avs_6$ con la de st_avsrsp_6 . Como el último par de la ruta de $set\_st\_avs_6$ y el primer par de la ruta de st_avsrsp_6 tienen el mismo identificador pero tiempos distintos, se suprime el último par de $set\_st\_avs_6$ porque no

se puede garantizar que su tiempo esté actualizado. La ruta resultante está formada por los siguientes pares (nodo, tiempo): $< 9ta_9$,$2ta_2$,$5ta_5$,$4ta_4\prime$,$6ta_6\prime$,$9ta_9>$. Al enviar el nodo 6 este mensaje, elimina, a su vez, el mensaje que ha empleado de $set\_st\_avs_6$ para formarlo y deja marcado inf_need_6 a *true*. La figura 5.45.h muestra la fase final de la detección del interbloqueo formado por los nodos 9, 2, 5, 4 y 6 . De todos los nodos que componen el ciclo, el nodo 9 es el único candidato que reúne la información necesaria para proclamarse detector y, en definitiva, víctima para resolver el interbloqueo. La información mencionada llega al nodo 9 a través del mensaje *AVS* procedente del nodo 6, que se ha descrito previamente. Dado que el identificador del primer y el último elemento de la ruta coinciden con el nodo receptor del mensaje (nodo 9), se tiene constancia de la existencia de un ciclo.

El orden establecido en la ejecución de las acciones que conducen a la detección se puede consultar en la tabla 5.8. El mecanismo de adquisión de identidades simuladas se ha demorado todo lo posible para que se aprecie cómo el algoritmo hace frente a esa situación. La detección del ciclo se ve ralentizada mientras la acción $rcvINF_7(3,m)$ no sea ejecutada. Por ejemplo, el mensaje *ALG* no es retirado por la acción $rcvALG_6(9,m)$ hasta que el nodo 6 asume una nueva identidad. Para ello, debe ejecutarse la acción $rcvINF_6(5,m)$ y, a su vez, previamente tiene que ejecutarse $rcvINF_7(3,m)$. Se puede observar gráficamente la presencia del mensaje *INF* dirigido al nodo 7 desde la figura 5.46.a hasta la 5.46.f. En lo que se refiere al mensaje *ALG*, éste aparece por primera vez en la figura 5.46.b y se mantiene hasta la figura 5.46.g.

El esquema de instancias de este ejemplo tiene la particularidad de que algunas de ellas fueron ya iniciadas en instantes anteriores para la detección del interbloqueo ya resuelto. En concreto, las instancias que lanzaron el nodo 7 y el nodo 5 retoman su actividad, manteniendo incluso la identidad (simulada) que presentaban en el ejemplo de la sección 5.2.2.4. También sigue su curso la instancia que controlaba el nodo 4 en representación del nodo 8. Con la identidad que adopta el nodo 4, $8ta_8$1-1, la instancia

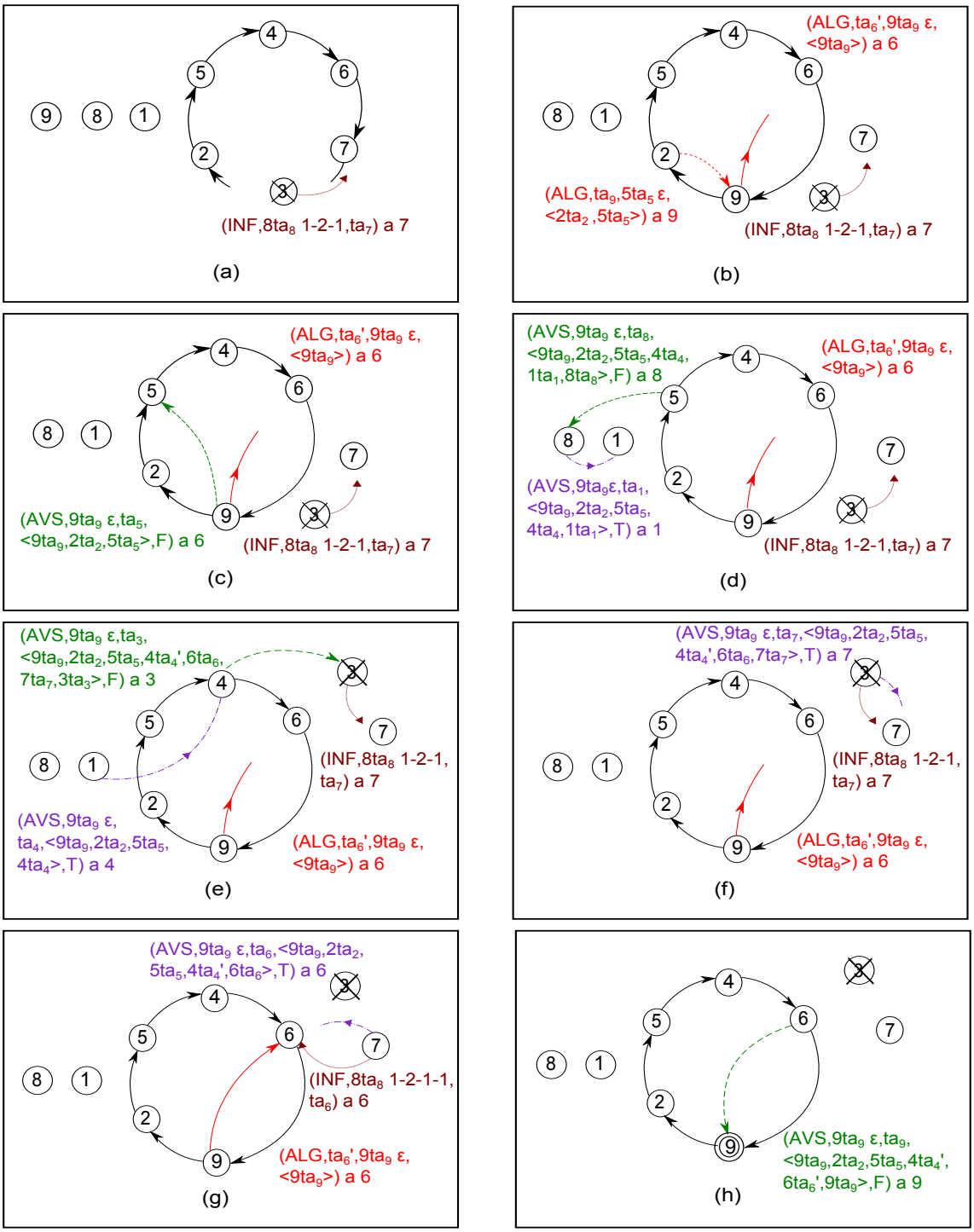

Figura 5.46: Detección de un nuevo ciclo tras la ruptura de un ciclo previo.

Inst. 1-nodo 7	**Dinamismo**	**Inst. 2-nodo 5**	**Inst. 3(*)-nodo 4**	**Inst. 4-nodo 9**
	$\text{SDA}_2(3, ta_3\prime)$ $\text{EDA}_7(3)$ $\text{SAA}_9(2)$ $\text{EAA}_2(9, ta_9\prime)$ $\text{SDA}_7(6, ta_6)$ $\text{EDA}_6(7)$ $\text{SAA}_6(9)$ $\text{EAA}_9(6, ta_6\prime)$			
		$\text{rcvALG}_9(2, m)$ $\text{rcvAVS}_5(9, m)$ sndAVS_5		initiate_9 firstAVS_9
	$\text{rcvAVS}_8(5, m)$ $\text{rcvAVS}_1(8, m)$			
			$\text{rcvAVS}_4(1, m)$ sndAVS_4	
Inst. 1(*)-nodo 7	$\text{rcvAVS}_3(4, m)$ $\text{rcvINF}_7(3, m)$			
Inst. 1(*)-nodo 6	$\text{rcvINF}_6(5, m)$			
	$\text{rcvAVS}_7(3, m)$			$\text{rcvALG}_6(9, m)$
$\text{rcvAVS}_6(3, m)$ sndAVS_6				 $\text{rcvAVS}_9(6, m)$
				Abort_9

Tabla 5.8: Secuencia de ejecución para un escenario dinámico con detección de mensaje *AVS* después de detectar y resolver un interbloqueo previo.

interactúa con el resto de instancias que progresan en el sistema.

Antes de que la resolución del interbloqueo anterior se hiciera efectiva, el nodo 3 transfiere su identidad para que sea el nodo 7 el que se haga cargo de dirigir su instancia. Como la evolución impuesta en el grafo de esperas lleva a que el nodo 7 deje de formar parte de la configuración considerada, éste tendrá que transferir el control de la instancia del nodo 3 al nodo 6 sin ni siquiera hacerlo suyo. La identidad simulada que le llega al nodo 7 tras el aborto del nodo 3 es $8ta_8$1-2 y la que finalmente adopta el nodo 6 para dirigir la instancia es $8ta_8$1-2-1.

Por otra parte, poco después de que el nodo 7 se desligue del grafo de esperas, un nuevo nodo, el nodo 9, pone en marcha la ejecución de una nueva instancia. Tras el

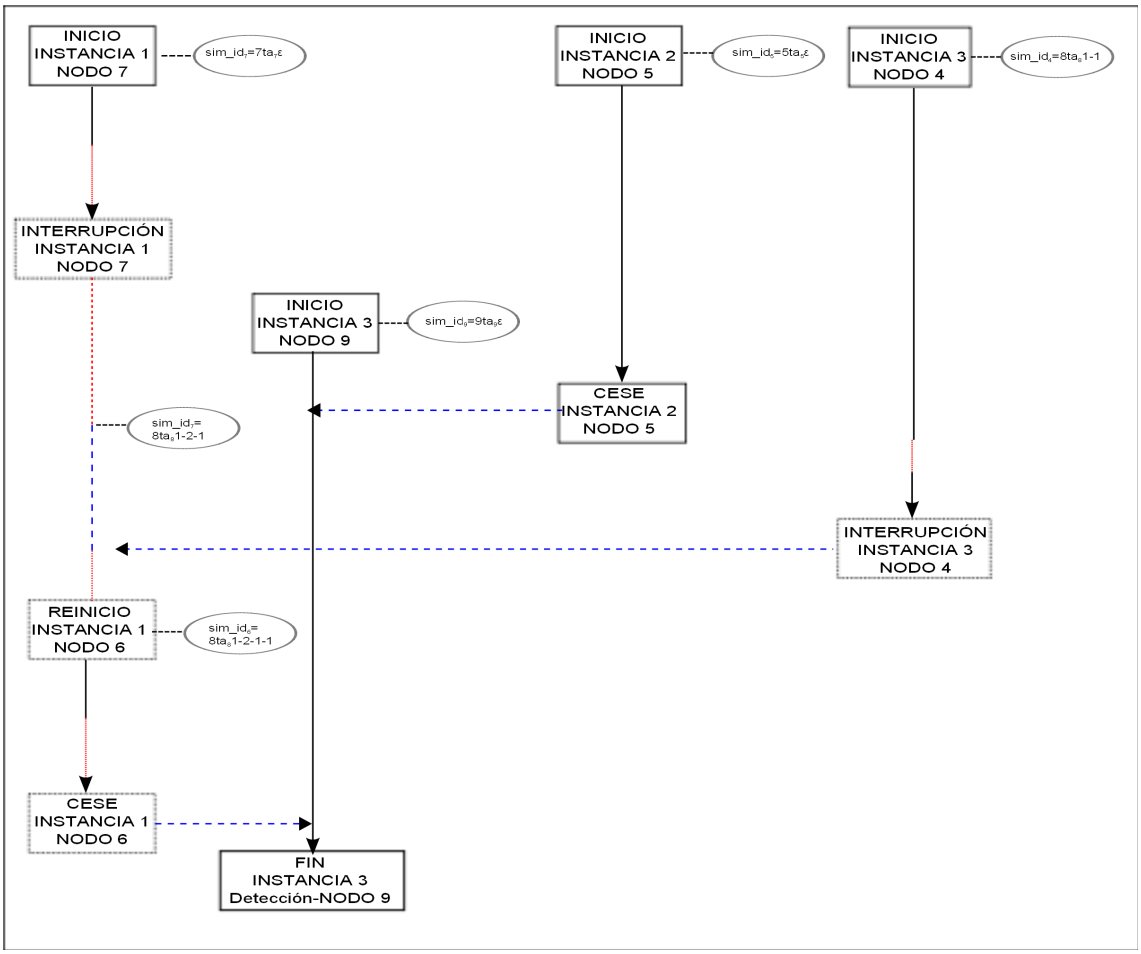

Figura 5.47: Diagrama de instancias del ejemplo de detección con *AVS* (caso dinámico).

intercambio de información entre el nodo 9 y el nodo 5, la instancia del nodo 5 deja de estar activa en el proceso de detección. Además, la instancia del nodo 4 que opera con la identidad $8ta_8$1-1 se da por terminada cuando en los mecanismos dinámicos del algoritmo la fuerzan a compararse con información residual del nodo 3 ($8ta_8$1-2). Así que, una vez finalizada la evolución del grafo y constituido el ciclo de esperas, hay dos instancias ejecutándose: la que simula el nodo 6 y la iniciada por el nodo 9. Sabiendo que la detección del interbloqueo la realiza el nodo con la identidad simulada mayor, la instancia del nodo 6 concluye dejando al nodo 9 como detector.

Capítulo 6

Prueba de corrección

En este capítulo se va a demostrar de una manera formal que el algoritmo propuesto cumple con el cometido para el que ha sido diseñado, o sea, se comprueba que funciona correctamente y cumple tanto el criterio de seguridad como de viveza que se definieron en el capítulo 3. Para ello, se muestra que el autómata S, que modela la solución propuesta, satisface la especificación del problema, E_0.

Para llevar a cabo las pruebas de corrección son necesarias una gran cantidad de propiedades. Incorporar las demostraciones de todas ellas dificulta sobremanera el seguimiento de las ideas fundamentales sobre las que se basan tanto la prueba del criterio de seguridad como la del criterio de viveza. Para facilitar la comprensión de la demostración del algoritmo, se ha decidido ubicar todas las propiedades que se necesitan en el apéndice B organizadas según sean propiedades relacionadas con el modelo del sistema, propias del criterio de seguridad y/o de viveza y, finalmente, las equivalencias entre las variables del medio y del algoritmo. En consecuencia, en las siguientes secciones solamente se presentan los enunciados de las propiedades más relevantes y se intercalan en las explicaciones de cómo se construyen, a partir de ellas, cada una de las pruebas de corrección.

6.1. Prueba de corrección del criterio de seguridad

En este apartado se comprueba que el algoritmo propuesto verifica el criterio de seguridad que se enunció en el capítulo 3. La demostración consiste básicamente en confirmar que, si existe un interbloqueo, el algoritmo fuerza el aborto de uno de los nodos del mismo.

Formalmente, el algoritmo, modelado mediante el autómata S, verifica el criterio de seguridad si cumple la precondición que incluye la acción $Abort_i$ del autómata S_0. De acuerdo con la teoría de Autómatas de Entrada/Salida, la existencia de un mapa de posibilidades de S en S_0 facilita la demostración porque dicho mapa garantiza que los comportamientos de S son un subconjunto de los comportamientos de S_0. Dado que la demostración no es inmediata es preciso formular una serie de propiedades que ayuden a probar que existe el mapa de posibilidades del autómata S en S_0.

La detección de un interbloqueo requiere la recepción de un mensaje ALG o un mensaje AVS en el que la ruta contenga una sucesión de nodos que describa un ciclo. En el caso de un mensaje AVS, el receptor del mensaje debe coincidir tanto con el primer nodo como con el último de la ruta que incluye. Además de asegurar que el nodo candidato recibe un mensaje con la ruta cumpliendo ese requisito, se tiene que asegurar que su identidad simulada es la mayor de entre todos los candidatos que haya en el ciclo. Por otra parte, la detección de un interbloqueo mediante un mensaje ALG implica que en su ruta ya está incluido el nodo receptor del mensaje. Los ciclos que se detectan de esta forma sólo tienen un nodo candidato y, obviamente, su identidad simulada es la mayor del ciclo. En consecuencia, señalar la víctima precisa analizar únicamente los nodos de la ruta del mensaje ALG.

Teniendo claras las condiciones en las que se puede producir la detección de un interbloqueo, a continuación se destacan las propiedades que permiten garantizar la correcta composición de las rutas necesarias en la detección y la elección del candidato

de mayor identidad simulada.

La propiedad B.2.55 señala que la ruta contenida en cualquier mensaje ALG o que esté almacenada en la variable *st_alg* de cualquier nodo cumple, en cualquier instante de ejecución del sistema, una serie de características. Considerando la existencia de un nodo i que tiene una relación de espera real con el nodo que le precede en la ruta analizada, se garantiza que todos los nodos anteriores al nodo i en la ruta son *dummy* y sus relaciones de espera están registradas correctamente en la ruta.

Aunque el estado del nodo destino del mensaje ALG puede ser cualquiera de los que se corresponden con el valor *blocked* en el medio, el emisor del mensaje tiene que ser el primer nodo de la ruta y el nodo destino debe pertenecer al conjunto de predecesores de este nodo. Respecto al primer nodo que aparece en una ruta almacenada en la variable *st_alg*, éste podría coincidir o no con el nodo que almacena la ruta. En la primera situación, se sabe que el estado del nodo es *dummy* y, por tanto, la ruta ha sido transmitida mediante un mensaje ALG al nodo que le precede si es que lo hubiere. Si, por el contrario, no hay coincidencia entre el nodo que almacena la información y el primer nodo de la ruta almacenada, se confirma que el estado del nodo que tiene almacenada la ruta es *candidate* y que es un predecesor del nodo que aparece en primera posición en la ruta. En este caso se asume que el nodo candidato deja de propagar la ruta a través de un mensaje ALG y pone en funcionamiento otros procedimientos para intercambiar información con otro candidato. El enunciado de la propiedad comentada es el siguiente:

> Sea $\{j,\ k\} \subseteq \mathcal{N}$, sea $sid \in T$, sea $p \in P$ tal que $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots$ $(n_m,\ t_m)$. Si $\exists\ i$ con $1 < i \leq m$ tal que $(n_{i-1},\ t_{i-1}) \in s.setPred_{n_i}$ entonces
>
> - $((ALG,\ s.ta_j,\ sid,\ p) \in s.channel(k,\ j) \Rightarrow \forall\ l < i\colon (n_l, s.ta_{n_l}) \in s.setPred_{n_{l+1}} \wedge s.status.alg_{n_l} = dummy \wedge (j,\ s.ta_j) \in s.setPred_{n_1} \wedge s.state_j = blocked \wedge n_1 = k) \wedge$

- $(s.st\_alg_j = (sid, p) \Rightarrow \forall\, l < i\colon (n_l, s.ta_{n_l}) \in s.setPred_{n_{l+1}} \wedge s.status.alg_{n_l} = dummy \wedge ((s.status.alg_j = dummy \wedge n_1 = j) \vee ((j,\ s.ta_j) \in s.setPred_{n_1} \wedge s.status.alg_j = candidate)))$.

La ruta contenida en un mensaje *ALG* o almacenada en *st_alg* contiene una serie de pares (nodo, tiempo) que puede coincidir total o parcialmente con las relaciones de espera existentes en el sistema en un instante de tiempo. El nodo de la ruta a partir del cual ya no hay correspondencia entre lo registrado en la ruta y las esperas reales del sistema es el que permite asegurar, según su estado y su identidad simulada, la evolución correcta del sistema.

En este punto de la demostración es necesario definir el concepto de camino o ruta borrada para simplificar la notación de las propiedades.

Definición 6.1.1. *Camino borrado entre dos nodos del grafo*, $ewait(i, j, p, s)$. Se dice que, según la información recopilada por el algoritmo en un estado $s \in states(S_0)$, hay una espera indirecta borrada entre un nodo $i \in \mathcal{N}$ y un nodo $j \in \mathcal{N}$ a través de una ruta $p = (n_1, t_1)(n_2, t_2) \ldots (n_m, t_m)$, almacenada en *st_alg*, *st_avsrsp* o incluida en mensajes *ALG* y *AVSRSP*, denotado como $ewait(i, j, p, s)$, si y sólo si:

- $n_1 = i \wedge n_m = j$
- $\forall\ k < m\colon (n_k, t_k) \in s.setPredToInf_{n_{k+1}} \wedge (n_k, t_k) \notin s.setPred_{n_{k+1}}$
- $\forall\ k < m\colon s.status.id_{n_{k+1}} = unknown \wedge s.t\_unk_{n_k} = t_k$

Si el nodo a partir del cual las esperas del sistema han sido eliminadas conoce su identidad simulada y su estado es *candidate*, *victim* o *active*, su identidad simulada es mayor o igual que la del campo *sid* del mensaje *ALG* o del mensaje almacenado en *st_alg*. En caso de que el nodo desconozca su identidad simulada, *status.id* es *unknown*, y si antes de activarse hubiera sido *candidate* con la variable *inf_need* a *true*, la identidad simulada es mayor o igual a la del mensaje correspondiente mensaje *ALG*. En esta siuación, el nodo está a la espera de un mensaje *INF* que le permita pasar a *known*. Ese mensaje *INF* procederá del nodo posterior a él en la ruta o

se irá propagando a través de los nodos que forman parte de la ruta pero ya no representan esperas reales.

Cuando el nodo es *dummy* o no conoce su identidad simulada (*unknown*) y antes de activarse era *dummy* (*inf_need* vale *false*), sucede algo similar al caso anterior. El nodo también espera recibir un mensaje *INF* con la identidad simulada que debe adquirir. Ese mensaje puede ser directo o llegar mediante sucesivas propagaciones de mensajes *INF* pasando por los nodos que conforman la parte *ewait* de la ruta del mensaje contenida en el *ALG* o en el almacén de este tipo de mensajes, *st_alg*. En estos casos, se cumple que el campo *sid* del mensaje *INF* es mayor que el campo del mensaje *sid* del mensaje *ALG* o de *st_alg*. En la propiedad B.2.56 del apéndice *B* se comprueban las características que cumple la ruta asociada a un mensaje *ALG* y se señala cuál es la identidad simulada mayor de los nodos que la componen. A continuación, se muestra el enunciado de dicha propiedad.

> Sea $\{j,\ k,\ x\} \subseteq \mathcal{N}$, sea $\{sid,\ sid\prime\} \in T$ y sea $\{p,\ p\prime\} \in P$ tal que $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_m,\ t_m)$. Si $(ALG,\ s.ta_j,\ sid,\ p) \in s.channel(n_k,\ n_j) \vee s.st\_alg_{n_j} = (sid,\ p) \wedge \exists\ i$ con $1 < i \leq m$ tal que $(n_{i-1},\ t_{i-1}) \in s.setPred_{n_i} \wedge (n_i,\ t_i) \notin s.setPred_{n_{i+1}} \Rightarrow$
>
> - $((s.status.alg_{n_i} \in \{candidate,\ victim,\ active\} \wedge s.status.id_{n_i} = known \wedge s.sim\_id_{n_i} \geq sid) \vee$
> - $(s.status.id_{n_i} = unknown \wedge s.inf\_need_{n_i} = true \wedge s.sim\_id_{n_i} \geq sid \wedge$
>   - $((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \vee$
>   - $(ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(x,\ last(p\prime).id))))$
>
> $\vee$
>
> - $((s.status.alg_{n_i} = dummy \vee (s.status.id_{n_i} = unknown \wedge s.inf\_need_{n_i} = false)) \wedge$

- $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
- $(ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(x,\ last(p\prime).id)) \wedge$
  - $((last(p).id \in nodes(p\prime) \wedge s.sim\_id_{last(p).id} \geq sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p) - last(p))$: $s.inf\_need_x = false \wedge s.inf\_need_{last(p).id} = true) \vee$
  - $(last(p).id \notin nodes(p\prime) \wedge sid\prime > sid \wedge \forall\ (x,\ t_x) \in visited\_nodes(p\prime)$: $s.inf\_need_x = false))))))$

A partir de estas dos propiedades, se asume que existe un ciclo si en un mensaje ALG aparece el nodo n_i en su ruta y este nodo es el receptor del mensaje ($n_j = n_i$). Al recibir el mensaje, basta con que el nodo n_i sea *candidate* para que se produzca la detección del interbloqueo. En esta situación, el resto de nodos del ciclo son *dummy* y han retransmitido el mensaje ALG, llegando a conformar la ruta del mensaje ALG que llega al nodo n_i. Como las esperas registradas en la ruta son reales hasta el nodo n_i (condición impuesta en ambas propiedades), el tratamiento del mensaje está garantizado ($m.ta = ta_{n_i}$). En el caso de que el nodo no fuera *candidate*, la última propiedad comentada pone de manifiesto que los mecanismos que lo convertirán en candidato y le harán adquirir una identidad simulada se están ejecutando.

A la hora de comparar las identidades simuladas de los nodos candidatos es importante tener en cuenta lo que dice la siguiente propiedad (ver demostración en el apéndice B, propiedad B.2.57). En ella se asegura que las identidades simuladas son únicas. En consecuencia, no se pueden encontrar dos nodos con la misma identidad simulada y, además, ningún mensaje *INF* puede contener en su campo *sid* la identidad simulada de un nodo en el mismo estado porque antes de propagar una identidad simulada se añade al último campo de dicha identidad el elemento *cont*, que lo hace diferente a todos los valores que ya existen y a los que un mismo nodo va a propagar

entre sus predecesores.

Sea $(id,\ t,\ \gamma) \in T$, se verifican simultáneamente las aserciones *A1* y *A2*.

A1: $\exists\ i \in \mathcal{N}$ tal que $s.sim\_id_i = (id,\ t,\ \gamma) \Rightarrow$

- *C1-1*: $\forall\ j \in \mathcal{N}$ tal que $j \neq i$ se verifica que $s.sim\_id_j \neq (id,\ t,\ \gamma)\ \wedge$
- *C1-2*: $\forall\ \{j,\ k\} \subseteq \mathcal{N}$, $\forall\ m \in M_{INF}$, si $m \in s.channel(j,\ k)$ entonces $m.sid \neq (id,\ t,\ \gamma)\ \wedge$
- *C1-3*: Si $\gamma = \alpha{\cdot}\beta$ tal que $\alpha \in \mathbb{N}^*$ y $\beta \in \mathbb{N}^{+} \wedge \exists\ j \in \mathcal{N}$ que verifica $s.sim\_id_j = (id,\ t,\ \alpha)$ entonces $s.cont_j > \mathit{first}(\beta)$.

A2: $\exists\ \{i,\ j\} \subseteq \mathcal{N} \wedge \exists\ m \in M_{INF}$ tal que $m \in s.channel(i,\ j)$ verificando $m.sid = (id,\ t,\ \gamma) \Rightarrow \forall\ \beta \in \mathbb{N}^*$:

- *C2-1*: $\forall\ k \in \mathcal{N}$ se verifica que $s.sim\_id_k \neq (id,\ t,\ \gamma{\cdot}\beta)\ \wedge$
- *C2-2*: $\forall\ \{k,\ l\} \subseteq \mathcal{N}$: $k \neq i \vee l \neq j$, $\forall\ m \in M_{INF}$, si $m \in s.channel(k,\ l)$ entonces $m.sid \neq (id,\ t,\ \gamma{\cdot}\beta)\ \wedge$
- *C2-3*: Si $\gamma = \alpha{\cdot}\delta$ tal que $\alpha \in \mathbb{N}^*$ y $\delta \in \mathbb{N}^{+} \wedge \exists\ k \in \mathcal{N}$ que verifica $s.sim\_id_k = (id,\ t,\ \alpha)$ entonces $s.cont_k > \mathit{first}(\delta)$.

Seguidamente, se analiza la detección de un interbloqueo cuando en el ciclo hay más de un candidato y es necesario que el candidato de mayor identidad simulada reciba un mensaje *AVS*, en el que la ruta describa una relación de esperas cíclica entre sus nodos. La formación de este mensaje *AVS* requiere la recepción de una serie de mensajes *AVS* y *AVSRSP* y la combinación de sus rutas que se almacenan en las variables *set_st_avs* y *st_avsrsp*, respectivamente. Por ello, es imprescindible enunciar una gran propiedad en la que estén perfectamente caracterizados y se controlen las rutas involucradas y la selección de la mayor identidad simulada de los candidatos que van intercambiando este tipo de mensajes.

Las rutas que aparecen en los mensajes *AVS* y *AVSRSP* y que están almacenadas en *set_st_avs* y *st_avsrsp* pueden haber perdido validez después posibles evoluciones dinámicas del sistema. En esta situación hay que asegurar que el algoritmo aporta mecanismos, bien para que esa información se adapte a las nuevas relaciones de espera entre nodos y se siga utilizando o bien, para que se pueda suprimir y deje de usarse. La siguiente propiedad permite controlar las características de las rutas consideradas suponiendo que existe un nodo n_i, a partir del cual los pares que le suceden en la ruta ya no describen una configuración de esperas reales. Debido a su extensión y para facilitar su comprensión, se va tratar de explicar por partes. El enunciado completo de la propiedad y su demostración se encuentran en el apéndice *B*, propiedad B.2.58. El antecedente de esta propiedad, común para las siguientes partes, es:

> Sea $\{n, n\prime\} \subseteq \mathcal{N}$, sea $\{t, t\prime\} \in \mathbb{N}$, sea $\{b, b\prime\} \in \{true, false\}$, sea $\{sid, sid\prime\}$ $\subseteq T$ y sea $\{p, p\prime\} \subseteq P$ tal que $p = (n_1, t_1)(n_2, t_2) \ldots (n_j, t_j)$. Si $((sid, t,$ $p, b) \in s.set\_st\_avs_{n_j} \vee (AVS, sid, t, p, b) \in s.channel(n, n_j) \vee (t, sid, p)$ $= s.st\_avsrsp_{n_1} \vee (AVSRSP, t, sid, p) \in s.channel(n, n_1)) \wedge \exists\ i$ con $1 <$ $i \leq j$ tal que $(n_{i-1}, t_{i-1}) \in s.setPred_{n_i} \wedge (n_i, t_i) \notin s.setPred_{n_{i+1}}$ entonces:

Inicialmente, se especifican una serie de características que presentan las rutas analizadas y que se cumplen en todas ellas hasta el nodo n_i que separa la parte de ruta que es real de la obsoleta. La parte de la ruta que es válida verifica que su primer nodo es candidato y que el resto de nodos, sin incluir el nodo n_i, mantiene su tiempo de activación desde que se incorporó a la ruta, conoce su identidad simulada ($status.id_{n_l} = known$) y su estado es *dummy* o *candidate*. Además, de ser alguno de ellos candidato, habrá intercambiado información con otros candidatos de la ruta y su identidad simulada será inferior al campo *sid* tanto de los mensajes *AVS* y *AVSRSP* como de las variables donde se almacenan estos mismos mensajes.

1. $s.status.alg_{n_1} = candidate \wedge$

2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i,\ t_i) \wedge wait(n_1,\ n_i,\ \gamma_1,\ s) \wedge$

3. $\forall\ l < i$: $s.status.id_{n_l} = known \wedge$

4. $\forall\ l$ tal que $1 < l < i$: $(s.status.alg_{n_l} = dummy \vee (s.status.alg_{n_l} = candidate \wedge sid > s.sim\_id_{n_l} \wedge s.inf\_need_{n_l} = true)) \wedge$

5. $\forall\ l < i$: $s.t\_unk_{n_l} = s.ta_{n_l} \wedge$

Otra particularidad que cumplen los mensajes almacenados en $set\_st\_avs_{n_j}$ es que todos los nodos que aparecen en su ruta se corresponden con esperas reales, esto es, $n_i = n_j$. El estado del nodo final de la ruta puede ser: *candidate*, *active*, *victim* o estar a la espera de conocer su identidad simulada, $s.status.id_{n_i} = unknown$. Por otra parte, el campo *sid* del mensaje almacenado coincidirá con la identidad simulada del primer nodo de la ruta y será mayor que la del último nodo de la ruta.

6. $(sid,\ t,\ p,\ b) \in s.set\_st\_avs_{n_j} \Rightarrow$

   - $(n_i = n_j \wedge$
   - $(s.status.alg_{n_i} \in \{candidate,\ active,\ victim\} \vee s.status.id_{n_i} = unknown) \wedge$
   - $(((sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.set\_st\_avs_{n\prime} \vee (AVS,\ sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.channel(n,\ n\prime)) \Rightarrow first(p\prime) \neq (n_1,\ t_1)) \wedge$
   - $(sid = s.sim\_id_{n_1} \wedge s.sim\_id_{n_1} > s.sim\_id_{n_j}) \wedge$
   - $s.nofirstAVS_{n_1} = false \wedge s.st\_avsrsp_{n_1} = NULL) \wedge$

Los mensajes *AVS* dirigidos al último nodo de su ruta cumplen que su campo *sid* coincide con la identidad simulada del primer nodo de su ruta. Además, se cumple que el primer nodo de la ruta ha enviado un mensaje *AVS* al recibir el mensaje *ALG* del candidato más proximo a él y, por tanto, su correspondiente $st\_avsrsp_{n_1}$

está vacío. La propiedad también niega la posibilidad de que, en el mismo instante de ejecución, exista un mensaje *AVS* propagándose o ya almacenado, cuyo primer nodo de la ruta sea exactamente el primer nodo de la ruta del mensaje *AVS* que se estudia. El nodo n_i puede que no conozca su identidad simulada, $s.status.id_{n_i} = unknown$, o hallarse en alguno de los siguientes estados: *candidate*, *active*, *victim*, *aborted* y *dummy*. Siendo *dummy*, el nodo n_i cumpliría que no se ha activado todavía si la ruta registra esperas no existentes o es un mensaje *AVS* que viaja en retroceso si la ruta completa representa esperas reales. En cualquiera de los dos casos el tiempo del nodo n_i que aparece en la ruta es su tiempo actualizado.

7. $(AVS,\ sid,\ t,\ p,\ b) \in s.channel(n,\ n_j) \Rightarrow$

   - $((s.status.id_{n_i} = unknown \vee s.status.alg_{n_i} \in \{candidate,\ active,\ victim,\ aborted\} \vee (s.status.alg_{n_i} = dummy \wedge s.ta_{n_i} = t_i \wedge ((b = true \wedge n_i = n_j) \vee (s.blocker_{n_i} = n_{i+1} \wedge n_i \neq n_j)))) \wedge$
   - $(((sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.set\_st\_avs_{n\prime} \vee (AVS,\ sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.channel(n,\ n\prime)) \Rightarrow first(p\prime) \neq (n_1,\ t_1)) \wedge$
   - $sid = s.sim\_id_{n_1} \wedge$
   - $s.nofirstAVS_{n_1} = false \wedge s.st\_avsrsp_{n_1} = NULL) \wedge$

Los mensajes *AVSRSP* almacenados en $st\_avsrsp_{n_1}$ o que se dirigen a este nodo se caracterizan por contar con un campo *sid* mayor que la identidad simulada del primer nodo de la ruta. Si la ruta se corresponde totalmente con las esperas existentes en el sistema ($n_i = n_j$), el estado del último nodo de esa ruta únicamente puede ser: *candidate*, *active*, *victim* o puede estar a la espera de conocer una nueva identidad simulada de otro nodo candidato al que ya no le unen esperas reales. En cualquier caso, la identidad simulada del último nodo será mayor o igual que la que procede del mensaje *AVSRSP*.

Cuando en la ruta hay registrados pares en su parte final, que no guardan relación con las esperas actuales del sistema ($n_i \neq n_j$), el nodo n_i puede encontrase en diferentes situaciones.

Si n_i conoce su identidad simulada ($status\_id_{n_i} = known$) siendo *candidate* con $inf\_need_{n_i}$ a *false*, *active* o *victim*, el valor de su identidad simulada es superior o igual al campo *sid* de la rutas referidas a un mensaje *AVSRSP*.

Por el contrario, si el nodo n_i no conoce su identidad simulada ($status\_id_{n_i} = unknown$) o aunque la conozca, está a punto de activarse y su $status\_id_{n_i}$ pasará a ser *unknown*, la propiedad garantiza que habrá un mensaje *INF* próximo a alcanzar el nodo n_i o que deberá propagarse a través de los elementos de una ruta que incluye, al menos, a los pares que no se asocian a esperas actuales (camino definido como *ewait*). La identidad simulada que viaja en ese mensaje *INF* es mayor que la que aparece en los mensajes *AVSRSP* o en su respectivo almacenamiento.

8. $((t,\ sid,\ p) = s.st\_avsrsp_{n_1} \vee (AVSRSP,\ t,\ sid,\ p) \in s.channel(n,\ n_1)) \Rightarrow$

- $(sid > s.sim\_id_{n_1} \wedge$
- $((n_i = n_j \wedge (s.status.alg_{n_i} \in \{candidate,\ active,\ victim\} \vee s.status\_id_{n_i} = unknown) \wedge s.sim\_id_{n_i} \geq sid) \vee$
- $((n_i \neq n_j \wedge n_i = n_1) \wedge$
  - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
  - $((ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,\ last(p\prime).id) \wedge sid\prime > sid)))) \vee$
- $((n_i \neq n_j \wedge n_i \neq n_1) \wedge$
  - $((s.status.alg_{n_i} \in \{candidate,\ active,\ victim\} \wedge s.status\_id_{n_i} = known \wedge s.inf\_need_{n_i} = false \wedge s.sim\_id_{n_i} \geq sid) \vee$

- $((s.status\_id_{n_i} = unknown \wedge s.inf\_need_{n_i} = true) \vee (s.status.alg_{n_i} = candidate \wedge s.inf\_need_{n_i} = true \wedge s.blocker_{n_i} = n_{i+1}) \wedge$
  1. $((sid,\ t,\ p - first(p)) = s.st\_alg_{n_1} \wedge$
     - $((((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid \leq s.sim\_id_{n_i})$ $\vee$
     - $(ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,$ $last(p\prime).id) \wedge sid \leq s.sim\_id_{n_i}))) \vee$
  2. $((sid,\ t,\ p - first(p)) \neq s.st\_alg_{n_1} \wedge$
     - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_i,\ n_{i+1}) \wedge sid\prime > sid > s.sim\_id_{n_i})$ $\vee$
     - $(ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,$ $last(p\prime).id) \wedge sid\prime > sid > s.sim\_id_{n_i}))))) \vee$
- $((s.status.alg_{n_i} = dummy \wedge s.blocker_{n_i} = n_{i+1}) \vee (s.status.id_{n_i} = unknown \wedge s.inf\_need_{n_i} = false) \wedge$
  1. $(((sid,\ t,\ p - first(p)) = s.st\_alg_{n_1} \wedge$
     - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
     - $(ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF,sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,$ $last(p\prime).id) \wedge$
       - $((last(p).id \in nodes(p\prime) \wedge s.sim\_id_{last(p).id} \geq sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p\prime) - last(p)) \wedge s.inf\_need_x = false \wedge s.inf\_need_{last(p).id} = true) \vee$
       - $(last(p).id \notin nodes(p\prime) \wedge sid\prime > sid \wedge \forall\ (x,\ t_x) \in visited\_nodes(\mathrm{p}\prime)$ $\wedge\ s.inf\_need_x = false)))) \vee$
  2. $((sid,\ t,\ p - first(p)) \neq s.st\_alg_{n_1} \wedge$
     - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$

$\circ$ $((ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF, sid\prime, s.t\_unk_{last(p\prime).id}) \in s.channel$
$(n, last(p\prime).id) \wedge sid\prime > sid)))))))))))$

Aplicando esta propiedad a la ruta que hace que un mensaje *AVS* sirva para detectar un interbloqueo, se confirma que el candidato receptor se transforma en *victim*. La ruta que representa un ciclo de esperas reales cumple que el destino de este mensaje *AVS*, n_j, coincide con n_i y, además, en la primera posición de la ruta también aparece el identificador de este mismo nodo. Por otra parte, esta misma propiedad permite asegurar que la identidad simulada del nodo que recibe el mensaje *AVS* de detección es la mayor de los candidatos del ciclo y es igual a la que se incluye en el mensaje *AVS*. De esta forma, no hay duda de que el resto de candidatos han quedado descartados de la elección de víctima porque han participado en la combinación de la ruta admitiendo que su identidad simulada es inferior. Si en el proceso de formación del ciclo las rutas que se van combinando recogen esperas que ya no existen en el sistema, la propiedad también indica la manera en la que la ruta se va corrigiendo y la forma en que se pueden propagar las identidades simuladas de candidatos obsoletos.

Por último, la existencia de un nodo víctima implica que existe un ciclo en el que todos los nodos conocen su identidad simulada, *status.id* contiene el valor *known*, y el resto de nodos del ciclo son *dummy* o *candidate*. El enunciado de esta propiedad aparece seguidamente y la demostración de la misma se puede revisar en el apéndice *B*, propiedad B.2.59.

$\exists\ i \in \mathcal{N}$ tal que $s.status.alg_i = victim \Rightarrow \exists\ p \in P$ tal que $wait(n, n, p, s)$
$\wedge\ \forall\ l \in nodes(p)$ se verifica que $s.status.id_l = known \wedge \forall\ l \in nodes(p) \setminus$
$\{i\}$: $s.status.alg_l = dummy \vee s.status.alg_l = candidate$.

Con estas propiedades ya se puede comprobar que el conjunto de comportamientos de S es un subconjunto de los comportamientos de S_0. Para ello, basta con en

demostrar la existencia de un mapa de posibilidades de S en S_0. Así que, en primer lugar, debe definirse una función que permita establecer ese mapa de posibilidades.

Definición 6.1.2. Función h: $states(S) \rightarrow 2^{states(S_0)}$. Se define la función h como la función que asocia a cada estado s del autómata S el subconjunto de estados $h(s)$ de S_0 que verifican:
$t \in h(s) \Leftrightarrow$

- $\forall\ i \in \mathcal{N}$: $t.set\_waiters_i = s.set\_waiters_i$
- $\forall\ i \in \mathcal{N}$: $t.bocker_i = s.blocker_i$
- $\forall\ i \in \mathcal{N}$: $t.t\_activ_i = s.t\_activ_i$
- $\forall\ i \in \mathcal{N}$: $t.state_i = s.state_i$

Tras definir la función h, se comprueba que h representa un mapa de posibilidades de S en S_0.

Propiedad 6.1.1. La función h es un mapa de posibilidades de S en S_0.

Demostración: Considerando la teoría de Autómatas de Entrada/Salida, para demostrar que h es un mapa de posibilidades de S en S_0 es necesario verificar:

- $Input(S) = Input(S_0)$ y $Output(S) = Output(S_0)$. Obvio a partir de las definiciones establecidas en las figuras 3.22 y 4.16.
- $t_0 \in h(\mathrm{s_0})$. La definición de los estados iniciales de ambos autómatas permite comprobar la condición:
  - $\forall\ i \in \mathcal{N}$: $t_0.set\_waiters_i = s_0.set\_waiters_i = \emptyset$
  - $\forall\ i \in \mathcal{N}$: $t_0.bocker_i = s_0.blocker_i = NULL$
  - $\forall\ i \in \mathcal{N}$: $t_0.t\_activ_i = s_0.t\_activ_i = 1$
  - $\forall\ i \in \mathcal{N}$: $t_0.state_i = s_0.state_i = active$
- Si s es un estado alcanzable de S, $t \in h(s)$ un estado alcanzable de S_0 y (s, π, $s\prime$) un paso de S, entonces se cumple:

- Si $\pi \in acts(S_0)$ entonces $(t,\ \pi,\ t\prime) \in steps(S_0)$ con $t\prime \in h(s\prime)$. Las acciones de S_0 modifican de igual modo las variables ya que las acciones de salida de ambos autómatas tienen en común todos los efectos que afectan a las variables que definen el mapa. Además, las acciones $\{\ \forall\ \{i,\ j\} \in \mathcal{N},\ \forall\ t \in \mathbb{N}\ StartAddArc_i(j),\ EndAddArc_i(j,\ t),\ StartDelArc_i(j,\ t),\ EndDelArc_i(j)\}$ presentan las mismas precondiciones tanto en S como en S_0. En el caso de la acción $Abort_i\ \forall\ i \in \mathcal{N}$, basta aplicar la propiedad B.2.59 para concluir que su precondición en S implica la correspondiente precondición en S_0. Por consiguiente, $(t,\ \pi,\ t\prime) \in steps(S_0)$ con $t\prime \in h(s\prime)$.
- Si $\pi \notin acts(S_0)$ entonces $t \in h(s\prime)$. En este caso la acción π se corresponde con una de las acciones internas de S y se comprueba fácilmente que ninguna de esas acciones modifica las variables $set\_waiters_i$, $blocker_i$ y t_activ_i. Sin embargo, la variable $status.alg_i$, que tiene a la variable $state_i$ como su equivalente en S_0, puede cambiar de valor al ejecutarse la acción π. Si $s.status.alg_i \in \{blocked,\ candidate\}$ y se ejecuta alguna de las acciones internas de S, se pueden alcanzar los siguientes estados $s\prime.status.alg_i \in \{candidate,\ dummy,\ victim\}$. Todos los valores de $status.alg_i$ mencionados se corresponden con el valor $state_i = blocked$ (propiedad B.3.4). Esto implica que $s.state_i = s\prime.state_i = blocked$ y, por tanto, la acción π tampoco modifica esta variable. En conclusión, $t \in h(s\prime)$.

■

Una vez establecido el mapa de posibilidades, se debe comprobar que los comportamiento de S son un subconjunto de los de S_0.

Teorema 6.1.1. *Seguridad. Los comportamientos del autómata S son un subconjunto de los comportamientos del autómata S_0, es decir, $behs(S) \subseteq behs(S_0)$.*

Demostración: Para demostrarlo es suficiente con asegurar que existe un mapa de posibilidades de S en S_0. Así que, teniendo en cuenta la propiedad 6.1.1, se confirma que la función h definida en 6.1.2 es un mapa de posibilidades de S en S_0 y, por tanto, el teorema es cierto. ■

En conclusión, dado que S verifica las mismas propiedades de seguridad que S_0 y se sabe que en S_0 solamente un proceso que forme parte de un interbloqueo puede abortar, en S también se cumple eso mismo. De esta forma, el algoritmo descrito por el autómata S cumple el criterio de seguridad.

6.2. Prueba de corrección del criterio de viveza

En esta sección se demuestra que el algoritmo verifica el criterio de viveza expuesto en el capítulo 3. El cumplimiento de este criterio permite asegurar que cualquier interbloqueo que aparezca en el sistema se resolverá en un tiempo finito. Para proceder a esta demostración se asume que el sistema definido por el autómata S cumple una propiedad de equidad débil sobre la partición del mismo. Esta propiedad garantiza que si hay una clase de la partición de S, $Part(S)$, que está habilitada continuamente, finalmente se ejecuta alguna de las acciones de esa clase.

Cuando un nodo candidato recibe un mensaje que permite garantizar que él es el nodo de mayor identidad simulada de entre todos los candidatos que conforman un interbloqueo, se asume que el interbloqueo ha sido detectado y el valor asociado a su variable *status.alg* pasa a ser *victim*. Dado que el interbloqueo es una situación estable durante el tiempo que se resuelve, el nodo que detecta el interbloqueo sigue formando parte de éste hasta que aborta. Esta idea queda enunciada en la siguiente propiedad (su demostración se puede localizar en la propiedad B.2.60 del apéndice B):

> Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S, $\alpha \in execs(S)$. $\forall\ i \in \mathcal{N}$ se verifica que $s_{z-1}.status.alg_i = victim \Rightarrow s_z.status.alg_i = victim \vee s_z.status.alg_i = aborted$.

Considerando este resultado y dado que las ejecuciones del sistema son equitativas, se puede afirmar que si el nodo i es el candidato que detecta el interbloqueo, entonces ese nodo será el que finalmente aborte. La demostración de esta afirmación aparece como propiedad B.2.61 del apédice B. A continuación se formula la propiedad.

> Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S, $\alpha \in execs(S)$. $\forall\ i \in \mathcal{N}$ se verifica que $\exists\ z_0$ tal que $s_{z_0}.status.alg_i = victim \Rightarrow \exists\ z_n > z_0$ tal que $s_{z_n}.status.alg_i = aborted$.

Así que, si se detecta un interbloqueo, éste siempre se resuelve. Para continuar la demostración del criterio de viveza, será suficiente con asegurar que todo interbloqueo que aparece en el sistema finalmente es detectado. Como la detección de un interbloqueo implica que el candidato de mayor identidad simulada reciba un mensaje en el que quede constancia de esa superioridad, habrá que comprobar que se genera y se envía ese mensaje.

Es importante recordar que cuando termina de formarse un ciclo en el sistema, los nodos pueden presentar diferentes estados, pero todos ellos son asimilables al estado *blocked* de la variable *state* del autómata S. Otra característica obligada de los nodos para que la detección del interbloqueo sea factible es que tengan conocimiento de su identidad simulada, o sea, que la variable *status.id* asociada a cada uno de los nodos del ciclo sea *known*. Como pone de manifiesto esta propiedad, ambos requisitos llegan a cumplirse (ver demostración de la propiedad B.2.66 del apéndice B):

> Sea $p \in P^+$ un ciclo en el estado s_z, $\mathcal{C}(p,\, s_z)$, $\forall\ n \in nodes(p) \wedge \forall\ z\prime > z$ se verifica que $s_{z\prime}.state_n = blocked \Rightarrow \exists\ z\prime\prime$: $z\prime\prime > z \wedge \forall\ n \in nodes(p)$, $s_{z\prime\prime}.status.id_n = known$.

La siguiente propiedad indica que, de todos los nodos candidatos existentes en un ciclo, sólo uno de ellos llegará a convertirse en víctima y, por tanto, detectar el interbloqueo y resolverlo. El enunciado de la propiedad se muestra seguidamente y su demostración se puede observar en la propiedad B.2.68 del apéndice B.

> Sea $p \in P^+$ un ciclo en el estado s_z, $\mathcal{C}(p,\, s_z)$. $\forall\ z\prime > z$: $\exists\ i \in nodes(p)$ tal que $\forall\ n \in nodes(p) \setminus \{i\}$: $s_{z\prime}.status.alg_n \neq candidate \Rightarrow \exists\ z\prime\prime > z$: $s_{z\prime\prime}.status.alg_i = victim$.

Asumiendo que sólo los nodos candidatos de un ciclo pueden ser los que detecten un interbloqueo y que únicamente uno se puede encargar de resolverlo, se pasa a

analizar cómo se identifica unívocamente al candidato detector de entre los posibles candidatos que haya. Si en el ciclo solamente hay un nodo candidato, la detección se realiza a través de un mensaje ALG. El candidato recibirá este mensaje y descubrirá que es el mensaje que él envió inicialmente (casos estáticos) porque en la ruta del mismo aparece su identidad y tiempo de activación, o bien que es un mensaje que retransmitió sin ser *candidate* y, tras una evolución dinámica del sistema, le vuelve a llegar siendo en ese instante un nodo candidato. La propiedad está demostrada en el apéndice B, propiedad B.2.67.

> Sea $p \in P^+$ un ciclo en el estado s_z, $\mathcal{C}(p, s_z)$. $\forall\ n \in nodes(p)$ se tiene que: $s_z.status.alg_n \neq victim \wedge \forall\ z\prime > z, \mathcal{C}(p, s_{z\prime}) \Rightarrow \exists\ z\prime\prime \geq z \wedge \exists\ i \in nodes(p)$: $s_{z\prime\prime}.status.alg_i = candidate \wedge \exists\ \{x, n\} \subseteq nodes(p)$: $\exists\ m \in M_{ALG}$ tal que $m \in s_{z\prime\prime}.channel(n, x) \wedge m.ta = s_{z\prime\prime}.ta_x \wedge (x, t) \in s_{z\prime\prime}.setPred_n \wedge m.path = \gamma_1 \cdot (i, t) \cdot \gamma_2$, siendo $\gamma_1, \gamma_2 \in P^*$.

Sin embargo, si en el ciclo hay al menos dos candidatos, la detección tendrá lugar después de que ellos hayan intercambiado información sobre las identidades simuladas de los candidatos que han conocido. Como se expone en la próxima propiedad, en cualquier ruta en la que haya varios candidatos, uno de ellos será el de mayor identidad simulada y se podrá establecer una ordenación de los candidatos que haya porque las identidades simuladas del sistema cumplen una relación de orden y no hay dos iguales.

> Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S. Sea $p \in P^+$ tal que $p = (n_1, t_1).(n_2, t_2)\ \ldots\ (n_j, t_j)$. Se verifica que $\exists\ z_0$: $\forall\ z \geq z_0$ tal que $wait(n_1, n_j, p, s_z) \wedge s_z.status.alg_{n_1} = s_z.status.alg_{n_j} = candidate \wedge \forall\ n \in nodes(p)$, $s_z.status.alg_n \neq victim$:
>
> - $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1} > max(\{s_z.sim\_id_{n_l}: 1 < l < j\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\}) \Rightarrow \exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se

verifica que $s_{z_n}.cand\_succ_{n_1} = (s_{z_n}.sim\_id_{n_j}.id,\ s_{z_n}.sim\_id_{n_j}.ta,\ nu)$, siendo $nu \leq s_{z_n}.sim\_id_{n_j}.nu \wedge$

- $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_j} > max(\{s_z.sim\_id_{n_l}: 1 < l < j\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\}) \Rightarrow \exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se verifica que $(s_{z_n}.sim\_id_{n_1},\ t,\ p,\ false) \in s_{z_n}.set\_st\_avs_{n_j}$ tal que $t_1 = s_{z_n}.ta_{n_1} \wedge t \in \mathbb{N}$.

Ese candidato es visto por el segundo candidato mayor como un nodo que, o bien le sucede, o bien le precede en las relaciones de espera registradas en la ruta considerada. Según como sea la relación de espera que hay entre ellos, el candidato con la segunda mayor identidad simulada de la ruta conocerá al mayor de los candidatos gracias al intercambio de mensajes. Si el candidato mayor es su sucesor, en su correspondiente variable *cand_succ* llegará a estar almacenada la identidad simulada del candidato mayor. Por el contrario, si el candidato de mayor identidad simulada de la ruta considerada precede o está esperando por el segundo candidato mayor, surgirá un mensaje *AVS* procedente del candidato mayor y dirigido al segundo mayor. Ese mensaje se procesará adecuadamente quedando almacenado en la variable *set_st_avs*. De entre toda la información que contiene el mensaje, la identidad simulada del candidato mayor es la más relevante para la continuidad de la búsqueda del candidato mayor en la ampliación de esa ruta. La comprobación de esta propiedad se puede encontrar como en el apéndice *B*, propiedad B.2.75.

Por tanto, teniendo en cuenta esta última propiedad, se va ampliando la ruta considerada hasta que coincida exactamente con la del ciclo que se tiene que detectar. Sabiendo que los candidatos llegan a contar con información de la identidad del candidato mayor que les precede o les sucede, según la configuración de esperas del ciclo, se alcanzará una situación en la que todo candidato dispondrá de información de un candidato que le sucede y de un candidato que le precede, siendo ambos mayores

que él. Dado que conoce la identidad simulada de esos dos candidatos, mediante un mensaje puede traspasar al menor de los dos la identidad del mayor. Esta manera de comparar las identidades simuladas involucra siempre a tres nodos candidatos. Cada vez que un nodo procede a señalarle a otro cuál de ellos es el de mayor identidad simulada, éste se autodescarta de la selección de víctima. Como el número de candidatos del ciclo es finito y en este proceso se va reduciendo el número de nodos que pueden seguir comparando sus identidades simuladas, llegará un momento en el que la comparación de identidades simuladas sea entre dos candidatos. De esos dos últimos candidatos, será el de menor identidad simulada el que tenga información, tanto en su variable *cand_succ* como en su *set_st_avs*, referida a la identidad simulada del mismo candidato (el mayor de todos). En esta última fase, el nodo con la segunda menor identidad simulada del ciclo enviará finalmente un mensaje *AVS* que transformará en víctima al receptor.

Sea cual sea el número de candidatos presentes en el ciclo, se puede asegurar que, de todos ellos, el que posea la mayor identidad simulada será el que llegue a convertirse en víctima, es decir, el que detectará el interbloqueo. Esta propiedad (ver demostración en apéndice B, propiedad B.2.76) se expresa formalmente como:

> Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S. $\forall\ n_i \in \mathcal{N}$, se verifica que $\exists\ p \in P^+$, $\exists\ z_0$: $\forall\ z \geq z_0$ se cumple $C(p, s_z) \wedge s_z.sim\_id_{n_i} = max(\{s_z.sim\_id_n$: $n \in nodes(p) \wedge s_z.status.alg_n = candidate\}) \Rightarrow \exists\ z_m \geq z_0$: $\forall\ z_n \geq z_m$, $s_{z_n}.status.alg_{n_i} = victim$.

En este punto de la demostración, en el que ya se ha comprobado que todos los ciclos se detectan y, además, que el nodo que detecta el interbloqueo aborta para resolverlo, se puede confirmar que cualquier interbloqueo que surja en el sistema modelado por S se resuelve en un tiempo finito. La propiedad que permite analizar este hecho se enuncia tal y como sigue. Su demostración se encuentra en el apéndice

B, B.2.77.

> Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S. $\forall\ n_i \in \mathcal{N}$, se verifica que: $\exists\ p \in P^+$ tal que $C(p,\ s_z) \Rightarrow \exists\ z_n > z_0$: $\pi_{z_n} \in \{Abort_{n_i}\}$ tal que $n_i \in nodes(p)$.

Teorema 6.2.1. *Viveza. El autómata S satisface la especificación* E_0.

Demostración: Como las ejecuciones equitativas del autómata S son las mismas que las de su módulo trivial de ejecuciones $Fair(S)$ y las del módulo de ejecuciones $E_0\prime$ son también las mismas que las de su módulo trivial de planes E_0, por definición bastará con demostrar que $Fair(S)$ satisface $E_0\prime$. Esto se consigue aplicando el teorema A.3.2, pero para ello primero es necesario comprobar que se cumplen las condiciones que hacen posible su aplicación.

Por un lado, el módulo de ejecuciones $E_0\prime$ se definió en el capítulo 3 asegurando que sus ejecuciones verificasen el siguiente predicado:

$\forall\ \mathcal{C} \in cycles(\mathcal{N})$ se verifica $Deadlock(\mathcal{C}) \hookrightarrow Resolution(\mathcal{C})$ donde,

$Deadlock(\mathcal{C}) = \{states(S)$ tal que $\mathcal{C} \in cycles(s)\}$ y $Resolution(\mathcal{C}) = \{\text{Abort}_i$ tal que $i \in nodes(\mathcal{C})\}$.

Por otra parte, como $fairexecs(Fair(S)) = fairexecs(S)$, la propiedad B.2.77 permite asegurar que las ejecuciones del módulo trivial de ejecuciones $Fair(S)$ verifican un predicado análogo al anterior:

$\forall\ \mathcal{C} \in cycles(\mathcal{N})$ se verifica $Deadlock\prime(\mathcal{C}) \hookrightarrow Resolution\prime(\mathcal{C})$ donde,

$Deadlock\prime(\mathcal{C}) = \{states(S)$ tal que $\mathcal{C} \in cycles(s)\}$ y $Resolution\prime(\mathcal{C}) = \{\text{Abort}_i$ tal que $i \in nodes(\mathcal{C})\}$.

Además, como $Fair(S)$ es un módulo de ejecuciones del autómata S y $E_0\prime$ es un módulo de ejecuciones del autómata S_0, por definición se cumple que $states(Fair(S)) = states(S)$ y $states(E_0\prime) = states(S_0)$. De este modo, por la propiedad 6.1.1, el mapa de posibilidades h definido en 6.1.2 relaciona los estados de $Fair(S)$ y $E_0\prime$, permitiendo asegurar que $h^{-1}(Deadlock(\mathcal{C})) = Deadlock\prime(\mathcal{C})$.

Finalmente, resulta obvio que $Resolution\prime(\mathcal{C}) = Resolution(\mathcal{C})$. Por lo tanto, teniendo en cuenta estos resultados, el teorema del apéndice asegura que $Fair(S)$ satisface $E_0\prime$. ■

Este resultado, según la teoría de Autómatas de Entrada/Salida, asegura que S verifica las mismas propiedades de viveza que E_0. De este modo, dado que en E_0 se resuelven en un tiempo finito todos los interbloqueos que aparecen, en S también se

cumple esa misma propiedad. Por lo tanto, queda probado que el algoritmo descrito mediante el autómata S cumple el criterio de viveza.

Capítulo 7

Estudio de complejidad

En los trabajos de algunos investigadores, como Singhal [55], se sugiere que el estudio formal de un algoritmo no debe limitarse sólo a demostrar su eficacia, sino que también debe considerar aspectos como su eficiencia y la existencia de limitaciones funcionales a la hora de implantarlo en un entorno real.

El análisis de la complejidad de los algoritmos permite la comparación entre ellos y, posiblemente lo que es más importante, conocer la escalabilidad de los mismos. Es muy frecuente plantearse la posibilidad de implementar un algoritmo para resolver un problema de mayor envergadura que aquel para el que estaba previsto. Un primer paso para determinar la viabilidad de este cambio consiste en estimar el incremento del tiempo y del número de recursos empleados por el algoritmo para completar su función. Estas estimaciones del tiempo y de otros recursos del sistema se conocen por el nombre de medidas de complejidad.

En este capítulo se pretende proporcionar un análisis detallado de la complejidad del algoritmo propuesto. Una vez que ha sido demostrado que el algoritmo cumple los objetivos para los que ha sido diseñado, esto es, resolver los interbloqueos que aparezcan en un sistema en un tiempo finito mediante el aborto de uno de los procesos interbloqueados, se va a analizar su rendimiento y se van a contabilizar los recursos que emplea para tal fin. Las medidas de complejidad que se consideran para este estudio

son principalmente dos: la complejidad en número de mensajes y la complejidad temporal.

7.1. Medidas de complejidad

Medir el rendimiento de un algoritmo distribuido supone un serio problema no en cuanto a calcular el espacio requerido por sus variables, sino por determinar el tiempo de ejecución del mismo [98]. En una estimación de la complejidad temporal de un algoritmo se suele asumir que las operaciones locales de un proceso no consumen tiempo [98]. Esto implica que la complejidad temporal de un algoritmo viene determinada por el número de mensajes que es necesario transmitir durante su ejecución. Aunque esta medida da una idea de lo que el algoritmo sobrecarga la red de comunicaciones, no tiene en cuenta que un algoritmo distribuido se puede estar ejecutando simultáneamente en mucho lugares del sistema y, por lo tanto, es muy probable que muchos de los mensajes necesarios en la ejecución se estén transmitiendo al mismo tiempo. En consecuencia, la complejidad temporal no dependerá linealmente del número de mensajes. Para estas situaciones en las que se analiza un algoritmo distribuido es más que conveniente determinar la complejidad mediante cadenas de mensajes. Así pues, la complejidad temporal viene dada por la mayor cadena de mensajes que llega a generar el algoritmo, en el que cada mensaje es consecuencia de uno anterior, y no por el número total de mensajes.

En resumen, en los sistemas en los que se supone un tiempo máximo para el retardo de los mensajes se contabilizará el total de mensajes generados por el algoritmo. Sin embargo, en sistemas en los que los mensajes podrían demorarse tiempos superiores a un retardo medio, se empleará como medida el tamaño de las cadenas causales de mensajes [98].

Aparte de estos inconvenientes, en los algoritmos de resolución de interbloqueos

surgen nuevos problemas. Estos algoritmos forman parte de un grupo que se denomina algoritmos de red dinámicos [50]. Este tipo de algoritmos se caracteriza porque su ejecución se realiza de acuerdo al estado del sistema y éste se modifica durante la ejecución del algoritmo, e incluso, puede modificarse por la propia acción del algoritmo, por ejemplo, tras la orden de aborto de un nodo. Según esto, no todos los mensajes que genera el algoritmo son útiles. Los mensajes que no influyen en la operación del algoritmo han sido creados porque existían esperas en el sistema que no forman parte de un ciclo. Contar el número de estos mensajes es una tarea bastante difícil ya que no se tiene control riguroso, ni de la aparición de las esperas en el sistema global, ni de la posibilidad de que algunas de esas esperas conformen una situación de interbloqueo.

La opción más frecuente a la hora de medir el número de mensajes en los algoritmos de resolución de interbloqueos consiste en tener en cuenta solamente los mensajes de detección y resolución de un interbloqueo establecido. A pesar de esta restricción pueden surgir mensajes que no están relacionados directamente con la detección/resolución en cuestión. De acuerdo con [56] y [118], la manera de proceder más habitual, a la hora de contabilizar el número de mensajes, es obviar aquéllos que no estén directamente relacionados con la detección del interbloqueo.

Teniendo en cuenta todo lo anteriormente explicado y, sabiendo que la estructura del interbloqueo tiene influencia en la complejidad resultante del proceso de resolución, se suele proporcionar la medida de complejidad correspondiente al caso más favorable (menor complejidad) y al menos favorable (máxima complejidad). Desde el punto de vista del sistema de comunicaciones es fundamental conocer cuántos mensajes genera el algoritmo de resolución de interbloqueo, pero si se considera la evolución del sistema, es más importante calcular el tiempo de latencia del interbloqueo. Si se cuenta el número de mensajes que son necesarios para eliminar el interbloqueo una vez originado, se obtendrá una medida de la latencia del algoritmo.

Debido a la falta de un criterio común en la manera de medir la complejidad de los

distintos trabajos resulta difícil comparar los algoritmos de detección y resolución de interbloqueos. Normalmente, para cada algoritmo se aporta la medida de complejidad que mejor se ajusta a su modo de operación. Para describir convenientemente la complejidad del algoritmo que se propone en esta tesis se estudiarán diversos casos. En lo que se denominarán casos estáticos, se va a suponer que el interbloqueo ya está formado y que no hay modificaciones en ninguna espera hasta que se resuelve el interbloqueo. Sin embargo, en los casos dinámicos se tratará la formación de un interbloqueo considerando la aparición y desaparición de las esperas del sistema. El estudio de los casos dinámicos, al recoger la dinámica del sistema, es más complicado que el de los casos estáticos, pero representa el caso más general.

Tanto en el algoritmo que se propone en esta tesis como en otras propuestas que también abordan la detección y resolución del interbloqueo, el estudio de la complejidad se centra en controlar el número de mensajes generados en su ejecución y que recorren el ciclo que representa la situación de interbloqueo a resolver. Este análisis, por tanto, establece una relación directa entre los mensajes enviados por el algoritmo y el tiempo que se necesita para resolver el problema, así como la relación de esos mismos mensajes con los recursos que se precisan para ello. Por consiguiente, las principales medidas de complejidad que se suelen presentar son:

- **Complejidad temporal.** Esta medida permite conocer la latencia del algoritmo en el proceso de resolución de un interbloqueo. Este valor temporal vendrá dado en función del número de nodos que están involucrados en el interbloqueo. El cálculo de la latencia se asimila al de la longitud de la secuencia causal más larga de mensajes enviados para resolver el interbloqueo [50]. Se denomina secuencia causal de mensajes a la cadena de mensajes en la que cada mensaje es consecuencia de la existencia del anterior.

- **Complejidad en número de bits.** Este valor proporciona una medida de la cantidad total de bits de información que se intercambian, mediante los mensajes que genera el algoritmo para resolver un interbloqueo [50].

Muchos de los algoritmos distribuidos, entre los que se incluyen los algoritmos de detección y resolución de interbloqueos, se caracterizan por el hecho de que su complejidad temporal depende del número de mensajes enviados en orden causal. El motivo de esta dependencia se debe a que el tiempo que transcurre desde que se envía un mensaje hasta que se recibe en su destino generalmente es mucho mayor que el tiempo que se emplea en el procesado de la información contenida en los mensajes.

A esto hay que añadir que el tiempo o retardo que supone la propagación de la información no es un factor fácilmente reducible, porque depende de las propiedades físicas de los materiales o de los medios usados como sistema de transmisión. Por otra parte, el retardo de los mensajes experimenta un crecimiento exponencial cuando se incrementa el volumen de tráfico de los canales de comunicación, [119]-[120]. En consecuencia, cuanto mayor sea la complejidad en número de bits del algoritmo, mayor retardo sufrirán los mensajes creados por el algoritmo y más tiempo costará resolver un interbloqueo dado. Una gran cantidad de trabajos publicados usan un tamaño fijo para sus mensajes: [55], [56], [68], [74], [70], [71], [112], [121] y [122], lo que hace posible en todos ellos asimilar la complejidad en número de bits a la complejidad en número de mensajes.

En el algoritmo presentado los mensajes cuentan con tamaños variables según sea su tipo y el número de nodos que conformen el interbloqueo. Esto implica que la complejidad en número de bits no coincidirá con la complejidad en número de mensajes. Dado que los mensajes generados por el algoritmo deben adaptarse además al protocolo de comunicaciones que se emplee en el sitema, se ha preferido proporcionar como medida de complejidad el número total de mensajes. Protocolos de comunicación habituales, como *UDP* [123] y *TCP* [124], organizan la información de los mensajes en

tramas con bits adicionales de señalización y de seguridad y para el cuerpo del mensaje dejan un espacio reservado que puede ser de longitud variable o fija. Esto implica que la transmisión de los mensajes va a depender más bien de la estructura de la trama de comunicación empleada y, por lo tanto, contabilizar el número de mensajes es una medida de complejidad que no supone ninguna limitación añadida al sistema.

7.2. Complejidad en número de mensajes

En esta sección se va a contabilizar el número de mensajes que se necesita para resolver un interbloqueo en el que intervienen n nodos. Analizando los posibles escenarios de ejecución se llegará a la conclusión de que este algoritmo presenta una complejidad lineal en número de mensajes, es decir, es un algoritmo de complejidad en número de mensajes $\mathcal{O}(n)$.

7.2.1. Casos estáticos

Generalmente, para determinar el coste de un algoritmo se supone un escenario estático en el que existe un interbloqueo (ciclo) de n nodos. Además, ninguno de los nodos que lo conforman ha iniciado la ejecución del algoritmo por lo que no existirán mensajes circulando en el sistema. Por otra parte, se asume que, desde el instante en que se forma el ciclo de esperas hasta el momento en que se resuelve el interbloqueo que representa ese ciclo, no tienen lugar modificaciones en el sistema.

El algoritmo proporciona un procedimiento distinto para la detección y resolución de interbloqueos, según haya un único nodo iniciador (candidato) del algoritmo en el ciclo o haya varios iniciadores. Esta diferenciación hace necesario realizar el cómputo del número de mensajes también en esos dos supuestos.

1. **Sólo un iniciador en el ciclo, $k = 1$**

Con un único candidato en el ciclo, resulta obvio que la única instancia activa del algoritmo es la que ese nodo lanza. La ejecución de esa instancia, por tanto, tiene que conducir a la detección y resolución del interbloqueo. El proceso comenzará en el instante en el que el nodo iniciador genere un mensaje ALG, proclamándose candidato a detector del interbloqueo. Como no hay ningún otro candidato en el ciclo, el mensaje ALG se difundirá por el ciclo y alcanzará de nuevo al nodo emisor de ese mensaje ALG. La propiedad B.2.68 del apéndice B expresa de modo formal la situación descrita en este caso. El total de mensajes que se precisa en esta detección es exactamente n: el mensaje ALG originado por el iniciador del algoritmo y $n-1$ mensajes ALG, derivados del mensaje ALG del iniciador, que retransmiten el resto de nodos del ciclo. Cabe mencionar en este punto que la fase de resolución no incrementa el número de mensajes contabilizado en la detección. Además, la simplicidad de este caso está motivada en cierta parte por el empleo de sólo un tipo de mensaje de los que están definidos en el algoritmo.

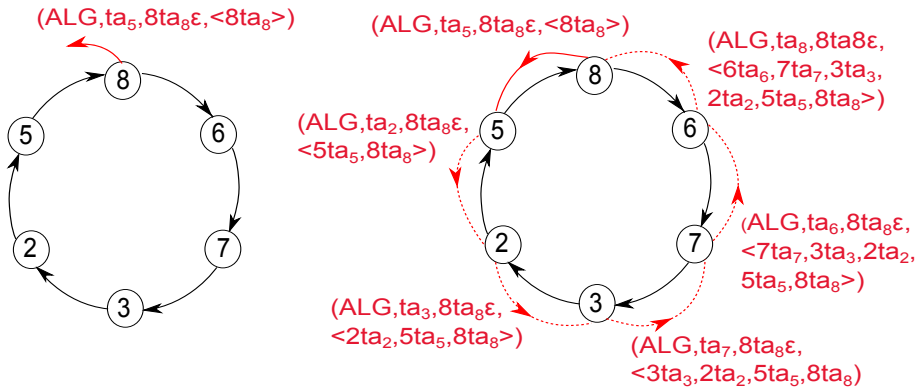

Figura 7.1: Detección en un ciclo estático con un iniciador ($k = 1$).

En el ejemplo de la figura 7.1, se muestra un ciclo formado por seis nodos. De esos seis nodos, tan sólo el nodo 8 inicia la ejecución del algoritmo. El mensaje ALG original (trazo continuo) es redirigido a través del ciclo hasta $n-1$ veces (mensajes

ALG de trazo discontinuo). En definitiva, en el ejemplo la detección del interbloqueo se logra gracias al envío de seis mensajes. Ese número de mensajes coincide con el número de nodos que forman el ciclo.

En los escenarios estáticos con un único iniciador no existe ni mejor ni peor caso de coste de mensajes. Por tanto, el algoritmo en esta situación presenta una cota inferior de coste $\mathcal{O}(n)$.

2. **k iniciadores en el ciclo, $1 < k \leq n$**

Al haber más de un iniciador en el ciclo, se debe considerar que se ejecutan simultáneamente k instancias del algoritmo. En esta situación el interbloqueo se detectará, como mucho, con $n+2k-2$ mensajes, siendo $k \leq n$. Este número de mensajes supone una cota superior y se corresponde con el caso estático menos favorable para el coste de comunicaciones del algoritmo. Esta complejidad se consigue gracias a la colaboración entre nodos para explorar el ciclo de esperas y al mecanismo empleado para decidir qué candidato entre dos posibles deja de participar en la detección. Dicho de otra manera, el método para reducir el número de instancias activas del algoritmo. Al final del proceso de detección sólo habrá una instancia activa del algoritmo. Esa instancia que perdura en todo el proceso de detección del interbloqueo es justamente la que inició el candidato de mayor identidad simulada del ciclo.

En un escenario estático como el que se está describiendo, la elección del candidato detector se reduce a comparar los identificadores de los nodos iniciadores y escoger el mayor de ellos. Por consiguiente, en un escenario estático donde se ha formado un interbloqueo sin que hayan circulado mensajes previamente, se puede establecer una correspondencia biunívoca entre los identificadores de los nodos y las identidades simuladas que presentan.

El funcionamiento del algoritmo se rige por las propiedades que se obtienen de las siguientes observaciones:

Observación 7.2.1. En un ciclo todos y cada uno de los nodos que lo componen envían un mensaje ALG a su predecesor en el ciclo.

Una vez que se ha formado un ciclo y k iniciadores han puesto en marcha sendas instancias del algoritmo, se puede afirmar que todos sus nodos han enviado sólo un mensaje ALG a su predecesor en el ciclo. De acuerdo con el código del algoritmo, sólo los nodos en estado *blocked-known* pueden enviar mensajes ALG en configuraciones estáticas. Cuando un nodo bloqueado envía un mensaje ALG, ya sea de manera espontánea o bien redirigiendo un mensaje ALG que previamente hubieran recibido, cambia el estado del nodo convirtiéndolo en un nodo *dummy* o *candidate*. Como no existe ninguna acción en el algoritmo que permita que un nodo *dummy* o *candidate* pase a ser de nuevo *blocked* sin una activación previa, se concluye que el nodo no podrá volver a enviar un mensaje ALG en el ciclo. Si tuviera lugar una activación previa del nodo, el ciclo analizado no existiría (caso estático).

Observación 7.2.2. Un mensaje almacenado en set_st_avs_i verifica que el candidato con la mayor identidad simulada se corresponde con el primer nodo de su ruta, $\text{sim\_id}_1 > \text{sim\_id}_i$. Sin embargo, un mensaje que esté almacenado en st_avsrsp_i verifica que el candidato de mayor identidad simulada que aparece en la ruta del mensaje es el candidato sucesor del nodo i. Este candidato sucesor coincide con el último candidato de la ruta que tenga el tiempo de activación correcto.

Las características que se aluden en este lema se derivan directamente de la propiedad B.2.58 del apéndice B. Esta propiedad es uno de los pilares en los que se apoya la prueba de seguridad del algoritmo.

Observación 7.2.3. Todos los candidatos de un ciclo, excepto el candidato que posee la mayor identidad simulada, envían sólo un mensaje AVS o AVSRSP a otro candidato del ciclo.

El candidato de mayor identidad simulada de un ciclo, durante la detección del interbloqueo, envía un mensaje ALG a su predecesor en el ciclo y recibe un mensaje AVS, que da por terminado el proceso de detección. Este mensaje AVS se formará después de que su identidad simulada haya sido comparada con la del resto de candidatos del ciclo. Como el candidato detector es el mayor candidato del ciclo (propiedad de seguridad), tan sólo podrá recibir un mensaje ALG de su único sucesor en el ciclo (modelo único recurso) y quedará habilitado el envío de un mensaje AVS. Por otra parte, el candidato detector de un ciclo podría llegar a tener mensajes en su correspondiente *set_st_avs*, pero se puede asegurar que ninguno de esos posibles mensajes provienen de un nodo del ciclo porque todos ellos son inferiores (con identidad simulada inferior). Como consecuencia directa de ser el nodo con mayor identidad simulada del ciclo, el nodo detector nunca podrá tener almacenado un mensaje en su st_avsrsp_i (propiedad B.2.51 del apéndice B).

El resto de candidatos del ciclo colaboran en la búsqueda del detector enviando mensajes AVS o $AVSRSP$ de acuerdo con las identidades simuladas que hayan quedado registradas en sus almacenes. Para poder generar cualquiera de esos tipos de mensajes, el nodo candidato tiene que tener información almacenada tanto en su *st_avsrsp* como en su $set\_st\_avs_i$. Aplicando la observación 7.2.1, la elección del candidato con la mayor identidad simulada está garantizada. Como el número de candidatos con identidad simulada menor es exactamente $k-1$, siendo k el número total de candidatos en el ciclo, se necesitan al menos $k-1$ mensajes ($AVS/AVSRSP$) para eliminarlos a todos ellos del conjunto de posibles detectores. Un nodo puede enviar ambos tipos de mensajes siempre y cuando tenga almacenado un mensaje en su *st_avsrsp*. El efecto de almacenar un mensaje en la variable *st_avsrsp* de un nodo puede producirse como consecuencia de la recepción de un mensaje ALG, procedente de un candidato superior, o de la recepción de un mensaje $AVSRSP$, notificando la existencia de un nodo mayor. Según este razonamiento, es obvio que el coste de

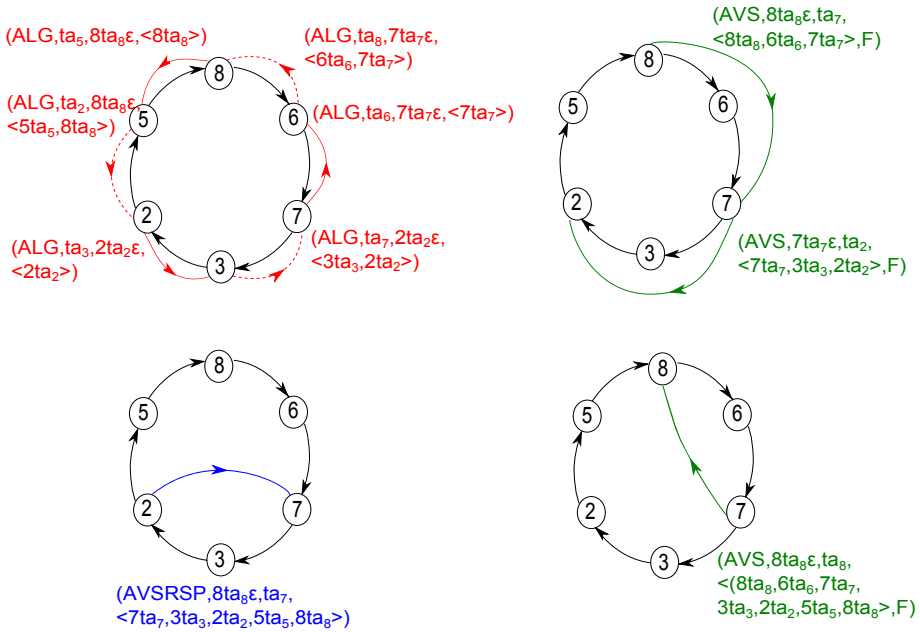

Figura 7.2: Detección en un ciclo estático con varios iniciadores ($k = 3$). Peor coste de comunicación.

comunicación depende directamente del número de candidatos que tengan mensajes almacenados en sus respectivos almacenes *st_avsrsp* como respuesta a un mensaje *ALG*. Por tanto, para determinar el número de mensajes necesarios en la detección de un interbloqueo en un ciclo de n nodos, de los cuales k son candidatos, se van a estudiar distintas configuraciones en las que hay un número diferente de variables *st_avsrsp* diferentes a *NULL* para los $k-1$ candidatos.

Peor caso de coste de comunicación. Este supuesto se caracteriza por la generación de $k-1$ mensajes *AVS* como respuesta a los *ALG* entre candidatos y el almacenamiento de sólo un mensaje *ALG* en el *st_avsrsp* del candidato que precede al futuro detector. Para rechazar los $k-1$ candidatos que tienen identidades inferiores a la del candidato detector se deben formar secuencialmente $k-2$ mensajes de tipo *AVSRSP*. Por último, el candidato con la segunda identidad simulada mayor del ciclo enviará el mensaje *AVS* final que señala al candidato detector. Sumando todos los

mensajes mencionados y añadiendo los n mensajes ALG que se difundieron por el ciclo, se obtienen $n+(k-1)+(k-2)+1$ mensajes, es decir, $n+2k-2$ mensajes para la detección del interbloqueo.

En el ejemplo de la figura 7.2, se representa un ciclo formado por seis nodos. De esos seis nodos, los nodos 8, 2 y 7 inician la ejecución del algoritmo. El sistema, por tanto, cuenta con tres instancias activas del algoritmo y el número de iniciadores es $k = 3$. Los mensaje ALG originales (trazo continuo) son redirigidos a través del ciclo hasta toparse con un nuevo candidato (mensajes ALG de trazo discontinuo). En la fase de exploración del ciclo se envían un total de seis mensajes de tipo ALG ya que cada iniciador se encarga de rastrear sin solapamientos una parte del ciclo. Por otra parte, dos de los tres candidatos del ciclo responderán a su candidato sucesor mediante un mensaje AVS y el otro candidato almacenará un mensaje en *st_avsrsp* ($k-1$ mensajes AVS). La única forma de progresar en la detección es combinar ese mensaje que está guardado en *st_avsrsp* con el mensaje AVS que reciba ese mismo nodo. Dado el ordenamiento de las identidades simuladas en este caso, surgirá de su combinación un mensaje $AVSRSP$ ($k-2$ mensajes) para que, una vez almacenado en *st_avsrsp*, se combine con el AVS que quedaba sin utilizar y se forme el AVS final (1 mensaje AVS de detección).

Mejor caso de coste de comunicación. En esta configuración sólo surge un mensaje AVS en el ciclo y $k-1$ mensajes almacenados en los *st_avsrsp* de $k-1$ candidatos. El número total de mensajes necesarios para detectar el interbloqueo será $n+(k-1)+1$, donde el primer sumando representa los mensajes ALG que envían todos los nodos del ciclo, $k-1$ son los mensajes AVS que se requieren para combinarse con los $k-1$ mensajes ubicados en *st_avsrsp* y, finalmente, se formará un mensaje AVS que permitirá la detección.

El ciclo que se muestra en la figura 7.3 cuenta con seis nodos, de los cuales tres son

iniciadores del algoritmo. Eso quiere decir que con 6 mensajes ALG se habrá conseguido recorrer todo el ciclo. La configuración de nodos de este caso sólo hace posible la formación de un mensaje AVS por parte de uno de los candidatos. Los otros dos candidatos a detector del ciclo almacenan mensajes en *st_avsrsp*. Así que uno de ellos se combinará con el AVS ya comentado y generará un nuevo mensaje AVS para unirse con el otro mensaje de *st_avsrsp*, que quedaba sin ser utilizado ($k-1$ mensajes AVS). De esta última combinación de mensajes surgirá el AVS final (1 mensaje AVS de detección).

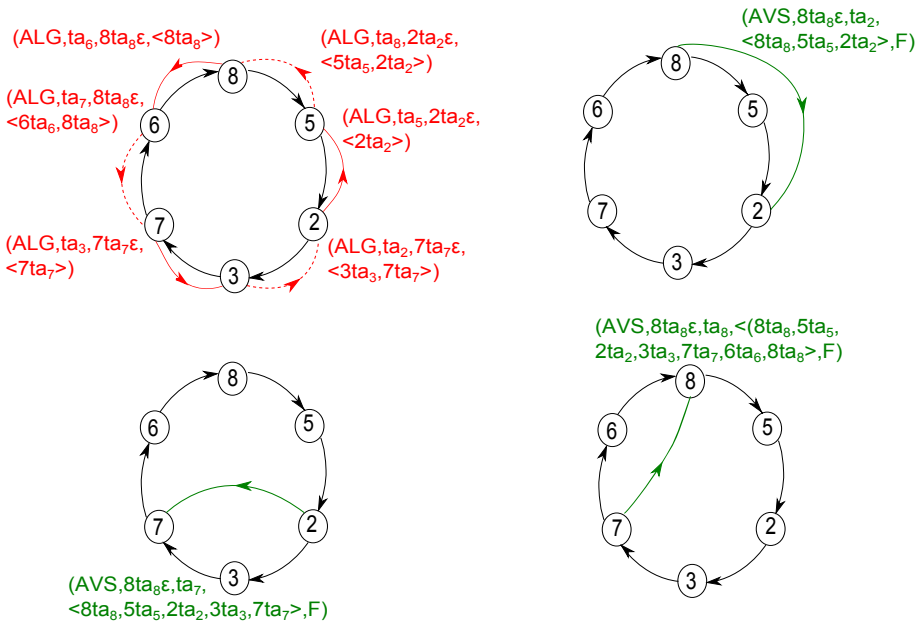

Figura 7.3: Detección en un ciclo estático con varios iniciadores ($k = 3$). Mejor coste de comunicación.

Cualquier otra combinación de mensajes AVS y de mensajes almacenados en *st_avsrsp* que se plantee, supone una ordenación diferente de los nodos candidatos en el ciclo y, por consiguiente, un coste intermedio limitado por el número de mensajes del peor caso y el número de mensajes del caso más favorable. En la tabla 7.1 se muestra un resumen del análisis realizado. Cabe destacar el dato asociado a $k = 2$ (iniciadores) que, aunque no se ha explicado separadamente, es una situación particular en la que

la resolución no depende de la localización de los iniciadores en el ciclo. Este es el motivo por el que aparece como único valor de coste el del mejor caso. Considerando todos estos resultados, se llega a la conclusión de que la complejidad en número de mensajes para casos estáticos es $\mathcal{O}(n)$, ya que el número de iniciadores, k, está acotado superiormente por el número de nodos, n, del que consta el interbloqueo.

k iniciadores	Mejor caso	Peor caso
1	n	$---$
2	$n+k$	$---$
$2<k\leq n$	$n+k$	$n+2k-2$

Tabla 7.1: Coste en número de mensajes para casos estáticos, según n.º iniciadores (k).

7.2.2. Casos dinámicos

En las situaciones comentadas anteriormente, los candidatos a detector del interbloqueo en el ciclo coinciden con los nodos iniciadores del algoritmo. Se puede comprobar fácilmente que cuando el número de candidatos es mucho menor que el número total de nodos que componen el ciclo, el número de mensajes empleados para la resolución del interbloqueo se reduce de manera ostensible.

El algoritmo que se ha desarrollado está dotado de mecanismos que permiten lograr la detección de un interbloqueo, incluso cuando los iniciadores del algoritmo no están incluidos en el ciclo que aparece en el sistema. Estudiar la complejidad en número de mensajes del algoritmo teniendo en cuenta estos casos es una tarea difícil por varios motivos. Uno de ellos es la falta de referencias en análisis de este tipo. En la literatura revisada, las medidas de complejidad que se ofrecen para algoritmos de detección y resolución de interbloqueos tienen que ver con escenarios estáticos en los que se considera ya formado el ciclo de nodos. En consecuencia, el cómputo

de mensajes sólo incluye los mensajes requeridos para la detección y resolución del interbloqueo establecida una relación de esperas determinada entre nodos.

Por otra parte, al ampliar la funcionalidad del algoritmo de detección y resolución a escenarios cambiantes, es lógico pensar que la contabilidad del número de mensajes se debe adaptar a esos nuevos escenarios. Esto implica que los mensajes que aparecen en el proceso previo a la formación del ciclo que se va a detectar son de gran interés para la medida del coste de este algoritmo. La medida del número de mensajes que se va a obtener será aparentemente peor que la que se ofrece en los casos estáticos, pero al tener en cuenta el funcionamiento real e íntegro del algoritmo, será una medida de rendimiento del algoritmo más fidedigna. Los aspectos relacionados con la complejidad serán precisamente un inconveniente a la hora de argumentar las bondades del algoritmo de esta tesis, ya que las comparaciones con otros algoritmos de detección y resolución existentes no se podrán efectuar en las mismas condiciones de funcionamiento.

Además, establecer una metodología para el estudio de la complejidad del algoritmo en casos dinámicos va a ser una tarea de gran complejidad, valga la redundancia. A pesar de tener definidas las acciones de formación y borrado de esperas entre nodos, y de tener limitado a uno el número de nodos por los que puede esperar cualquier nodo del sistema (modelo de asignación de único recurso), las evoluciones que conducen a la formación de un ciclo son innumerables. Para reducir el número de evoluciones posibles del sistema, el algoritmo minimiza el número de mensajes en los casos dinámicos imponiendo ciertas limitaciones. Esas limitaciones son:

- La cancelación del retroceso de mensajes *AVS* cuya ruta está formada por un uńico nodo. Ese mensaje no aporta información.

- La ordenación impuesta para el procesamiento de mensajes: un mensaje *AVS* en retroceso no podrá ser tratado por un nodo *unknown* hasta que se haya

procesado el mensaje *INF* correspondiente que transforma el estado del nodo a *known*. De esta forma, se evita la retransmisión de mensajes cuya información puede estar a punto de pasar a ser anticuada y debería ser revalidada.

- La reutilización de mensajes *AVS* recibidos o almacenados por un nodo que ha cambiado su identidad simulada. En vez de eliminar completamente esa información del sistema, el nodo la transformará en un mensaje *AVSRSP* dirigido a otro nodo que la encuentre todavía coherente con su función en el sistema.

- El borrado de mensajes *INF* que no tienen relevancia o no es posible procesarlos. Así se evita el envío de mensajes innecesarios.

El método empleado para calcular el coste de comunicación del algoritmo en casos dinámicos analizará, en primer lugar, las posibles formaciones de un ciclo a partir de una cadena simple de nodos. Para estudiar los mecanismos dinámicos que aporta el algoritmo, se propondrán situaciones en las que el algoritmo sea iniciado por un número creciente de nodos. Además, se va a forzar que en el proceso de formación del ciclo se eliminen esperas y se cree una nueva espera que cierre el ciclo con nodos que pertenecían a la cadena inicial.

Cadena de nodos con un iniciador que evoluciona a un ciclo con un iniciador simulado $(\boldsymbol{k} = 1)$

La información del iniciador se propaga hasta el último elemento (predecesor). Si las esperas que suceden al iniciador se eliminan no surge ningún mensaje. Si se cierra el ciclo uniendo el último elemento de la cadena con el iniciador, la detección del ciclo conlleva el mismo número de mensajes que el caso estático de $k = 1$, esto es, n. Si se borran esperas más allá del iniciador y se cierra el ciclo con una espera, que une el último nodo de la cadena con un predecesor activado del iniciador, el cálculo del número de mensajes requiere contabilizar una serie de mensajes que en el caso estático

no estaban presentes. Un ejemplo de esta evolución dinámica se puede observar en la figura 5.14 del capítulo 6.

Para contar con exactitud los mensajes asociados a esta situación, se debe conocer el número de esperas rotas entre el iniciador de la cadena y el nodo predecesor que inicia la espera que finalmente forma el ciclo. En lo que sigue, se llamará m_1 al número de las esperas rotas entre el iniciador y el nodo de la cadena que formará parte del ciclo. De acuerdo con la evolución dinámica que se ha indicado, se debería añadir $2m_1$ mensajes a los n mensajes necesarios para detectar el ciclo final de n nodos (caso estático). De los $2m_1$ mensajes nuevos, m_1 mensajes se corresponden con mensajes ALG, que se propagaron por las esperas eliminadas, y los otros m_1 mensajes son mensajes *INF*, que propagan la identidad del iniciador de la cadena conforme se van borrando las esperas de la cadena. El último mensaje *INF* es el encargado de que el nodo de la cadena, que aún sigue conectado al resto, adquiera la identidad del iniciador. A pesar de que este nodo se comporta como un nodo *dummy* desde que deja pasar la información del iniciador, cuando se activa y recibe el mensaje *INF*, se convierte en un nodo *candidate* que hará las veces del iniciador. En el ciclo final, el nuevo estado del nodo de la cadena permite que el algoritmo prosiga su ejecución y no quede paralizado a expensas de tener que invocar otra instancia del algoritmo. El coste en número de mensajes de esta formación básica de un ciclo es $n+2m_1$. Por tanto, el algoritmo en esta situación dinámica presenta un coste $\mathcal{O}(n+m)$, donde m hace referencia al número no acotado de esperas rotas necesarias para que el ciclo se forme.

Cadena de nodos con más de un iniciador que evoluciona a un ciclo con sólo un iniciador simulado ($\boldsymbol{k} = 1$)

Todos las configuraciones agrupadas en este caso se caracterizan por el hecho de que los iniciadores iniciales de la cadena han intercambiado toda la información

posible, antes del proceso dinámico que elimina las esperas que los unen a la cadena de nodos de partida. Ninguno de esos iniciadores iniciales formará parte del ciclo resultante.

1. *Con 2 iniciadores iniciales.* Cuando se borran esperas de la cadena de manera que queda sin los nodos iniciadores, se identifican dos grupos de esperas rotas, las que se localizan entre el iniciador y el nodo dummy que pertenece a la cadena inicial, m_1, y las que hay entre los dos iniciadores, m_2. Tanto el valor m_1 como el valor m_2 deben ser conocidos si se desea acotar el número de mensajes que existen en esta evolución dinámica.

   Si el iniciador que primero se activa es el de mayor identidad (simulada), la comunicación entre iniciadores no habrá generado ningún mensaje directo que, al suprimirse las esperas entre ellos, precise ser rescatado. En esta situación, por tanto, hay que considerar en el primer tramo de esperas rotas m_2 mensajes *ALG* y m_2 mensajes *INF*. En el segundo tramo de esperas rotas se tendrán en cuenta m_1 mensajes *ALG* y m_1 mensajes *INF*. En total, la detección del ciclo requiere de $n+2m_1+2m_2$ mensajes.

   En el caso de que el primer iniciador que se activa sea el menor de los iniciadores presentes en la cadena, es necesario contabilizar, además de los mensajes m_1+m_2 *ALG* y los m_1+m_2 *INF*, el mensaje *AVS* que hayan intercambiado los iniciadores y los m_2-1 *AVS* en retroceso que se originan. El número de mensajes *AVS* en retroceso es una unidad menos que m_2, porque el algoritmo tiene implementado un mecanismo que evita enviar mensajes *AVS* en retroceso que contengan un sólo nodo en su ruta. Un mensaje de estas características no va a aportar nada de información al nodo destino ya que el nodo de la ruta es él mismo. En resumen, la cantidad total de mensajes que son necesarios para la detección de este ciclo es $n+2m_1+3m_2$, donde n son los mensajes que coinciden con la detección del

ciclo estáticamente.

2. *Con* 3 *iniciadores iniciales*. Las evoluciones dinámicas de una cadena de nodos, que conducen a un ciclo con un único iniciador simulado, comienzan a ser más numerosas. Dependiendo de la comparación de las identidades (simuladas) de los iniciadores y de los mensajes que surjan al establecer esas comparaciones, el coste de comunicación oscilará, para 3 iniciadores, entre $n{+}2m_1{+}3m_2{+}3m_3{+}1$ (2 mensajes AVS directos) y $n{+}2m_1{+}2m_2{+}2m_3$ (ningún mensaje AVS directo entre iniciadores). En estas expresiones se supone que m_3 son las esperas rotas entre el primer y el segundo iniciador que se activan; m_2 es el número de esperas que se elimina entre el segundo y el tercer iniciador de la cadena que se activan y, por último, m_3 son las esperas que desaparecen desde el tercer iniciador activado y el nodo de la cadena que permanece y formará parte del ciclo, siendo su iniciador simulado.

A continuación, la tabla 7.2 muestra un resumen del coste en mensajes de distintas evoluciones dinámicas, incluidas las explicadas hasta el momento. En todas ellas el ciclo se detecta por un iniciador simulado ($k = 1$) y mediante un mensaje ALG. Teniendo en cuenta todos los datos registrados en la tabla, se concluye que la complejidad en número de mensajes para este escenario dinámico básico es $\mathcal{O}(n{+}m)$, donde n son los nodos interbloqueados y m se corresponde con las esperas eliminadas entre nodos iniciadores en el proceso de formación del interbloqueo.

Cadena de nodos que evoluciona a un ciclo con un iniciador simulado y otro real ($\boldsymbol{k} = 2$)

En la configuración que seguidamente se analiza, la cadena de nodos incluye dos iniciadores. Uno de ellos formará parte del ciclo final a detectar y el otro quedará desvinculado de la cadena de nodos después de haber comunicado su identidad a sus

Número de iniciadores iniciales activados	Número de mensajes
1	$n+2m_1$
2	$n+2m_1+3m_2$ $n+2m_1+2m_2$
3	$n+2m_1+3m_2+3m_3+1$ ... $n+2m_1+2m_2+2m_3$
4	$n+2m_1+3m_2+3m_3+3m_4+1$ ... $n+2m_1+2m_2+2m_3+2m_4$
5	$n+2m_1+3m_2+3m_3+3m_4+3m_5+1$ ... $n+2m_1+2m_2+2m_3+2m_4+2m_5$

Tabla 7.2: Coste de comunicación, según número de inicadores iniciales activados, para ciclos con un iniciador simulado, $k = 1$.

predecesores. Pensando que el ciclo final tendrá dos iniciadores, uno real y otro simulado, se asume que el número de mensajes necesarios para detectarlo será superior a los señalados en el caso estático equivalente, $n+k$. A la hora de contabilizar los mensajes que conducen a la detección del interbloqueo, deben diferenciarse dos casos según el estado del iniciador simulado antes de formar parte del ciclo final.

1. *Estado previo del iniciador simulado: dummy.*

Suponiendo que se rompen m_1 esperas entre el iniciador de la cadena, que se activa en primer lugar, y el nodo *dummy* de la cadena que pertenecerá al ciclo final, se computan $2m_1$ mensajes (m_1 mensajes *ALG* y m_1 mensajes *INF*), que deben sumarse a los $n+k$ mensajes previstos para $k = 2$ en el caso estático equivalente. Los mensajes *INF* son los responsables de propagación de la identidad simulada del iniciador, que queda desvinculado de la cadena de nodos original y, por tanto, del ciclo final.

Al alcanzarse el último nodo de la cadena que se activa, éste adquiere la identidad simulada y se convierte en un iniciador simulado. Un ejemplo concreto de la detección que se plantea en este caso aparece representada gráficamente en la figura 5.21 del capítulo 6.

Si el iniciador real del ciclo es mayor que el iniciador simulado, el iniciador real envía un mensaje *AVS* directo al iniciador desaparecido que, a la vez que los mensajes *INF*, debe llegar retrocediendo nodo a nodo hasta el iniciador simulado. El retroceso del mensaje *AVS* supone también m_1 mensajes. Como el mensaje *AVS* directo puede asimilarse al *AVS*, que en el caso estático con $k = 2$ serviría para comparar los dos candidatos a detector del ciclo, no se añadirá a la cuenta de mensajes del caso dinámico. En cuanto el iniciador simulado reciba el *AVS* en retroceso, se generará el mensaje *AVS* detector que también está incluido en el coste obtenido para el caso estático. La cantidad total de mensajes creados es $(n+k)+3m_1$.

Aunque el número de mensajes necesario para la detección es similar al caso anteriormente descrito, cuando el iniciador real del ciclo es menor que el iniciador simulado, surge otra ordenación de mensajes. Un vez cerrado el ciclo, el iniciador simulado responderá con un *AVS* directo al iniciador real y posteriormente, el iniciador real formará un nuevo mensaje *AVS* directo con destino el iniciador activado. El mecanismo dinámico que hace llegar a este *AVS* al iniciador simulado se pondrá en funcionamiento, siendo el último retroceso equivalente al mensaje de detección del caso estático.

En la tabla 7.3 se proporcionan los valores de coste en número de mensajes de otras evoluciones que se agrupan en el caso que se acaba de explicar. Los registros de la tabla que aparecen para 2 o más nodos iniciadores activados se corresponden con los valores máximo y mínimo de coste. Como la disposición de esos iniciadores, que no formarán parte del interbloqueo final detectado, determina el número de mensajes generados en la detección, es obvio que pueden obtenerse costes intermedios a partir de

3 iniciadores activados. Recuérdese que el número de esperas rotas, m_i, hace referencia a las esperas eliminadas en el tramo i, siendo el primer tramo roto de la cadena ($i = 1$), el que va desde el último iniciador activado hasta el iniciador simulado, el segundo tramo roto ($i = 2$), el que se localiza entre el penúltimo y el último iniciador activado, el tercer tramo ($i = 3$) entre el antepenúltimo y el penúltimo iniciador activo y así sucesivamente.

A la vista de los costes registrados en todas la situaciones planteadas, se deduce que la complejidad en número de mensajes de la formación dinámica de un interbloqueo como la descrita en este apartado es también $\mathcal{O}(n+m)$. El número de mensajes obtenido en cada caso es $(n+k)+m$, donde n es el número de nodos del interbloqueo, k el número de iniciadores, reales y simulados, que aparecen en el ciclo final y m representa las esperas rotas entre nodos iniciadores que han desaparecido para poder formarse el ciclo. A pesar de que m no se puede acotar superiormente, k lo está por el número de nodos del ciclo detectado. Esto implica que la complejidad, se puede expresar como $\mathcal{O}(n+m)$.

2. *Estado previo del iniciador simulado: candidate* con *inf_need = true*.

En el caso que se describe a continuación, el iniciador simulado del ciclo final debe haber enviado un mensaje, esto es, la variable *inf_need* asociada a este nodo tiene que tener el valor *true*. Esto implica que el mensaje enviado ha surgido tras la comparación de la identidad (simulada) de un iniciador que le sucede y otro que le precede. Se asume entonces que la cadena de nodos, que permite explicar esta situación, debe contar al menos con tres nodos iniciadores. Dos de ellos pertenecerán al ciclo resultante, uno con una nueva identidad simulada (iniciador simulado) y otro con su identidad real. El resto de iniciadores deberá desligarse de la cadena para que aparezca el interbloqueo deseado. Considerando la cadena de nodos más simple (3 nodos iniciadores), se llega fácilmente a establecer el coste de comunicación de esta

Número de iniciadores iniciales activados	Número de mensajes
1	$(n+k)+3m_1$
2	$(n+k)+3m_1+3m_2$ $(n+k)+3m_1+2m_2$
3	$(n+k)+3m_1+3m_2+3m_3+1$ ... $(n+k)+3m_1+2m_2+2m_3$
4	$(n+k)+3m_1+3m_2+3m_3+3m_4+1$ ... $(n+k)+3m_1+2m_2+2m_3+2m_4$
5	$(n+k)+3m_1+3m_2+3m_3+3m_4+3m_5+1$ ... $(n+k)+3m_1+2m_2+2m_3+2m_4+2m_5$

Tabla 7.3: Coste de comunicación, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*dummy* previamente) y un iniciador real ($k = 2$).

evolución dinámica. Al número de mensajes de la misma configuración estática, $n+k$, se deben agregar obligadamente m_1 mensajes ALG y m_1 mensajes INF.

Para comprender la evolución dinámica que se explica en este apartado se pueden consultar los ejemplos de ejecución 5.28 y 5.34 del capítulo 6. En estas dos situaciones aparecen mecanismos como el borrado de mensajes INF y la reconversión de mensajes $AVSRSP$ que permiten reducir el cómputo global de mensajes INF y AVS en la detección de este tipo de interbloqueo.

La tabla 7.4 proporciona los valores de coste en la cadena simple (un iniciador activado) diferenciando los casos en los que el detector se corresponde con el iniciador que adquiere una identidad simulada o con el iniciador de la cadena que no se ha activado. Los siguientes valores hacen referencia a una cadena de nodos con cuatro

iniciadores en el que dos de ellos se activan y no forman parte del ciclo final. Como puede observarse, hay una gran variedad de costes según qué nodo sea el detector y, en caso de que el detector sea el iniciador simulado, se registran valores diferenciados según la ubicación del iniciador del que se hereda la identidad simulada. Aunque se podría ampliar esta tabla incrementando el número de iniciadores activados, con los datos registrados basta para deducir que la complejidad en número de mensajes es también $\mathcal{O}(n+m)$.

Número de iniciadores iniciales activados	Número de mensajes	Detector
1	$(n+k)+3m_1+1$	iniciador real
	$(n+k)+2m_1+3$	iniciador simulado
2	$(n+k)+3m_1+3m_2+1$	iniciador real (mejor caso)
	$(n+k)+3m_1+3m_2+2$	iniciador real (peor caso)
	$(n+k)+2m_1+2m_2+3$	iniciador simulado (mejor caso)
	$(n+k)+3m_1+2m_2+4$	iniciador simulado (del activado más alejado)
	$(n+k)+2m_1+3m_2+3$	iniciador simulado (del activado más próximo)

Tabla 7.4: Coste de comunicación, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*candidate* previamente) y un iniciador real ($k = 2$).

Cadena de nodos con más de un iniciador que evoluciona a un ciclo con un iniciador simulado y dos reales ($\boldsymbol{k} = 3$)

En este apartado se estudia la evolución de una cadena de nodos con más de tres iniciadores, que resulta en un ciclo gracias a la activación de varios iniciadores. Pensando que el ciclo final tendrá tres iniciadores, dos reales y otro simulado, se asume que el número de mensajes necesarios para detectarlo será superior a los señalados

en el caso estático equivalente, $n+k$ (mejor caso) o $n+2k-2$ (peor caso). Para contabilizar los mensajes que conducen a la detección del interbloqueo se diferencian a su vez dos casos que consideran el estado del iniciador simulado antes de formar parte del ciclo final.

1. *Estado previo del iniciador simulado: dummy.*

Dado que el nodo hasta el que se eliminan las esperas de la cadena inicial era *dummy*, éste hereda una identidad simulada de alguno de los iniciadores iniciales activados y se convierte en un candidato que competirá por ser el detector del ciclo final. El iniciador simulado junto con los otros dos candidatos, que no se han desligado de la cadena inicial de nodos al formarse el ciclo, serán los 3 iniciadores que intervendrán en la detección del ciclo. Al igual que pasaba en el escenario estático con 3 iniciadores, en la configuración resultante también afectará la ordenación de los candidatos. Esto implica que según cuáles sean las relaciones de espera entre ellos, se podrá obtener diferente número de mensajes correspondientes al mejor y al peor caso de coste de comunicación.

En la tabla 7.5 aparecen registrados los valores de coste obtenidos para cadenas de nodos en las que se incrementa el número de iniciadores iniciales que se activan. A patir de dos iniciadores activos, surgen más de dos secuencias diferentes de mensajes porque esos iniciadores antes de activarse intercambian nuevos mensajes que hay que contabilizar. Además, en este caso se pondrá en marcha el mecanismo de retroceso de mensajes *AVS* y, según el número de esperas que separan al nodo *dummy* del iniciador del que hereda su identidad simulada, la cantiad de mensajes *AVS* será diferente. Generalizando los datos recogidos en la tabla, se puede concluir que el número de mensajes que se precisa en la detección de un interbloqueo originado de la forma descrita es $(n+k)+m$ en el caso más favorable y $(n+2k-2)+m$ en el peor caso, siendo k el número de iniciadores del ciclo final y m el número de esperas rotas entre

los iniciadores activados y el iniciador simulado. En resumen, la complejidad de la evolución dinámica analizada es $\mathcal{O}(n+m)$.

Número de iniciadores iniciales activados	Número de mensajes
1	$(n+k)+3m_1$ (mejor caso) $(n+2k-2)+3m_1$ (peor caso)
2	$(n+k)+3m_1+2m_2$ ... $(n+2k-2)+3m_1+3m_2$
3	$(n+k)+3m_1+2m_2+2m_3$ ... $(n+2k-2)+3m_1+3m_2+3m_3+1$

Tabla 7.5: Coste de comunicación, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*dummy* previamente) y dos iniciadores reales ($k = 3$).

2. *Estado previo del iniciador simulado: candidate* con *inf_need = true*.

En la evolución que se describe seguidamente, el ciclo final contará con tres iniciadores, uno simulado y otros dos presentes en la cadena original de nodos. A diferencia del apartado anterior, el iniciador simulado ya era iniciador antes de formarse el ciclo. Por tanto, este iniciador simulado ha intercambiado información con otros inicadores que se han activado para la formación del ciclo y/o iniciadores que no se han desvinculado de la cadena original de nodos. El hecho de que el iniciador simulado fuera candidato antes de formarse el interbloqueo hace que el número de mensajes necesario para la detección sea diferente según el valor de la identidad simulada con la que se compara con el resto de iniciadores del ciclo.

La tabla 7.6 muestra los valores calculados para diversas configuraciones de iniciadores en la cadena original de nodos. Se puede observar que hay un número diferente

de mensajes según cuál sea la mayor identidad simulada de los iniciadores presentes en el ciclo. El caso de un iniciador activado proporciona dos números de mensajes distintos, el que se obtiene cuando el candidato de mayor identidad simulada es un candidato presente en el ciclo y en la cadena de nodos inical y en el que el iniciador simulado es el mayor porque la identidad simulada del iniciado activado que ha adquierido es la mayor del ciclo. En estos cálculos se ha tenido en cuenta tanto los mensajes *ALG* e *INF*, que se propagaron por las esperas rotas entre los inicadores activados y el iniciador simulado, como los mensajes *AVS* en retroceso, que se difunden también por esas esperas, y los posibles mensajes *AVSRSP* reconvertidos. Para el caso de dos iniciadores activados es evidente que se incrementan las posibilidades de ordenación, tanto de los iniciadores reales como de los activados y, por lo tanto, se puede hallar gran diversidad de valores. En cualquier caso, el número de mensajes se puede expresar como $(n+k)+m$ y la complejidad en este tipo de evoluciones con $k = 3$ es $\mathcal{O}(n+m)$.

Nótese que el sumando $2k$ que aparece en las tablas 7.5 y 7.6 es el mismo que el de los casos estáticos, siendo 2 un valor constante que no depende del número de iniciadores, k.

Se podría seguir con el análisis de complejidad en número de mensajes para escenarios dinámicos incluyendo nuevas configuraciones. Por ejemplo, se podrían estudiar formaciones de ciclos en los que se combinan cadenas de nodos de los distintos tipos descritos en esta sección, ciclos formados por cadenas de nodos en forma de y (ejemplo de ejecución 5.40 del capítulo 6), tridente,... o ciclos reconstruidos a partir de cadenas de nodos resultantes de una resolución previa (ejemplo de ejecución 5.46 del capítulo 6). Para todas esas formaciones será válida la conclusión que se ha alcanzado en todas las evoluciones planteadas, esto es, la complejidad en número de mensajes de los escenarios dinámicos que resuelve el algoritmo propuesto es $\mathcal{O}(n+m)$. Esto implica que el número de mensajes necesario para detectar un interbloqueo en una

Número de iniciadores iniciales activados	Número de mensajes	Detector
1	$(n+k)+3m_1+1$	iniciador real (mejor caso)
	$(n+k)+3m_1+3$	iniciador simulado (mejor caso)
	$(n+2k-2)+3m_1+1$	iniciador real (peor caso)
	$(n+2k-2)+3m_1+3$	iniciador simulado (peor caso)
2	$(n+k)+3m_1+3m_2+1$	iniciador real (mejor caso)
	$(n+k)+3m_1+2m_2+4$	iniciador simulado (mejor caso)
	$(n+2k-2)+3m_1+3m_2+2$	iniciador real (peor caso)
	$(n+2k-2)+3m_1+3m_2+1$	iniciador simulado (peor caso)

Tabla 7.6: Coste de comunicación, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*candidate* previamente) y dos iniciadores reales ($k = 3$).

evolución dinámica es proporcional al número de nodos del ciclo, n, y al número de esperas entre nodos que iniciaron una instancia del algoritmo y que se han eliminado para la formación del ciclo.

7.3. Complejidad temporal

En esta sección se proporciona otra medidad de complejidad que tiene que ver con el tiempo de propagación de todos los mensajes involucrados en el proceso de resolución del interboqueo. La latencia del algoritmo para detectar un interbloqueo también es lineal incluso si se consideran secuencias causales de mensajes. Sabiendo que el cierre del interbloqueo no lleva a la creación inmediata del mensaje que detectará el interbloqueo, se van a estudiar los diferentes escenarios estáticos y dinámicos que se pueden dar.

7.3.1. Casos estáticos

Para los casos estáticos se supondrá, como en la sección anterior, que ninguno de los nodos que forman el ciclo habrá iniciado la ejecución del algoritmo y, por tanto, no existirán mensajes circulando en el sistema. Por otra parte, se asumirá que, desde el instante que se forma el ciclo de esperas hasta el momento en que se resuelva el interbloqueo que representa ese ciclo, no tendrán lugar modificaciones en el sistema.

Según el número de iniciadores del algoritmo y la distribución de los nodos bloqueados en el ciclo, se obtendrán diferentes tiempos de latencia del interbloqueo. A continuación, se muestra el análisis realizado que incluye inicialmente el caso más sencillo en el que hay un único iniciador ($k = 1$) para, finalmente, llegar a proponer el caso genérico de un ciclo de n nodos con k iniciadores, siendo $1<k\leq n$.

Caso $\boldsymbol{k} = 1$

Cuando en el ciclo sólo existe un candidato (iniciador), la latencia es $n{\cdot}T$, donde T es el retardo de comunicación y n el número de nodos que conforman el ciclo. El mensaje ALG inicial que genera el candidato debe retransmitirse por el resto de nodos del ciclo. Hasta que el mensaje ALG no alcanza su nodo destino, este nodo no puede dirigir el mensaje ALG a su predecesor. El ciclo que se representa en la figura 7.1 muestra con claridad el número de mensajes causales, o sea, la complejidad temporal, que conlleva este caso. Por tanto, el tiempo de latencia obtenido es lineal con el número de nodos del ciclo, $\mathcal{O}(n)$.

Caso $\boldsymbol{k} > 1 \wedge \boldsymbol{k} = \boldsymbol{n}$

Las configuraciones que aquí se consideran son aquellas en las que el número de nodos del ciclo coincide con el de iniciadores del algoritmo de detección. Todas las situaciones que se engloban en este caso presentan como latencia $(n{+}1){\cdot}T$, siendo T el retardo de comunicación de los mensajes intercambiados en el proceso de detección y

n el número de nodos (iniciadores) que conforman el ciclo. Por consiguiente, el tiempo de latencia obtenido es proporcional al número de nodos del ciclo, $\mathcal{O}(n)$. Para $k = n = 2$, la secuencia causal de mensajes está formada por un único mensaje *ALG*, un mensaje *AVS* en el que quedan comparados los dos candidatos iniciadores y un nuevo mensaje *AVS* que sirve como detección del ciclo. Al contabilizar sólo un mensaje *ALG* se presupone que los dos iniciadores están lanzando sus mensajes de iniciación al mismo tiempo y, tanto su propagación como su recepción, están solapadas en el tiempo.

Conforme se incrementa el número de nodos (iniciadores) del ciclo, n, se observa que, a pesar de que deben intercambiarse más mensajes para lograr la detección del ciclo, la latencia del algoritmo viene determinada por la misma expresión, esto es, $(n+1)\cdot T$. Esta conclusión se alcanza después de haber analizado las configuraciones que establecían el mejor y el peor caso del estudio de la complejidad en número de mensajes.

Si se considera el mejor caso, los mensajes *ALG* de los iniciadores se propagan y reciben a la vez, seguidamente se suceden los mensajes *AVS* necesarios para reducir el número de instancias del algoritmo a solamente una y se concluye, finalmente, con un mensaje *AVS* de detección. El computo de estos mensajes viene dado por la relación $1+(k-1)+1$ que, en su forma simplificada, es $k+1$ ó $n+1$. En este punto cabe destacar el comportamiento causal de los mensajes *AVS*. A diferencia de los mensajes *ALG* de los iniciadores, las comparaciones entre los distintos candidatos no puede solaparse en el tiempo.

En lo que respecta a las configuraciones del denominado peor caso, la secuencia causal de mensajes incluye el envío de mensajes *ALG*, la comparación de inciadores mediante mensajes *AVS*, la sucesión de mensajes *AVSRSP* y la detección del interbloqueo realizada por un mensaje *AVS*. Al contabilizar los mensajes que conforman la cadena causal de ejecución, se considera que los mensajes *ALG*, al igual que los

mensajes *AVS*, se solapan en el tiempo. Sin embargo, los mensajes *AVSRSP* se forman secuencialmente, siendo la causa que los origina el efecto de la recepción del mensaje anterior. La expresión que permite contar todos los mensajes de esta cadena de ejecución viene dada por $1+1+(k-2)+1$ o, lo que es lo mismo, $k+1$ ó $n+1$.

Caso $\boldsymbol{k > 1 \wedge k < n}$

Para abordar este caso, se van a distinguir dos tipos de ciclos diferentes: aquéllos en los que el número total de nodos, n, es divisible por el número de nodos iniciadores k, y los ciclos en los que n no es divisible por k. Cuando la relación de n y k es un número entero, es posible encontrar configuraciones balanceadas (los nodos bloqueados se distribuyen uniformemente entre los nodos iniciadores). En cambio, si n/k no es un valor entero, la distribución de los nodos bloqueados no puede ser uniforme. El motivo principal de hacer este análisis según la razón n/k consiste en que la disposición de los nodos bloqueados en el ciclo influye directamente en el tamaño de la cadena causal que se debe ejecutar para la detección del mismo. A continuación, se presentan el estudio de la latencia para $k = 2$ y para $k > 2$ en las dos situaciones mencionadas.

1. $\boldsymbol{n/k \in \mathbb{Z} \wedge k = 2}$

   - Al ser n/k un número entero es posible una distribución equilibrada de los nodos no iniciadores en el ciclo. Este reparto equitativo de los nodos bloqueados entre los nodos candidatos implica que en la cadena causal de ejecución van a poder solaparse en el tiempo, tanto los mensajes *ALG* de los iniciadores como los mensajes *ALG* que redireccionan los nodos bloqueados hasta alcanzar otro iniciador. Por otro lado, el mensaje *AVS* de comparación de iniciadores surgirá como respuesta a uno de los mensajes *ALG* redireccionados que alcanza, finalmente, al otro iniciador del ciclo. Una vez recibido este mensaje *AVS*, será generado el mensaje *AVS* que se encargará de detectar el ciclo. El total

de mensajes que componen esta cadena causal de ejecución se corresponde con $n/k+(k-1)+1$. Por lo tanto, la latencia obtenida es $(n/k+k)\cdot T$, donde T es el retardo de comunicación de los mensajes intercambiados en el proceso de detección.

- Otras configuraciones en las que varía la longitud de la cadena causal de mensajes son las que tienen localizados en el ciclo a los nodos bloqueados, o bien todos como predecesores del candidato (iniciador) mayor, o bien todos como predecesores del candidato menor. Si todos los nodos bloqueados se sitúan entre los candidatos, de manera que sean predecesores del iniciador con mayor identidad, sólo será posible el solapamiento de los mensajes ALG de los iniciadores. El resto de mensajes ALG que deben redireccionarse tendrán que crearse secuencialmente. Por otra parte, el mensaje AVS que aparece tras la comparación de los dos iniciadores puede adelantarse porque no precisa de la información que redirigen los mensajes ALG. De acuerdo con este comportamiento, no habrá que contabilizar ningún mensaje AVS de los que reducen el número de instancias de ejecución, pero seguirá siendo necesario contar con el mensaje AVS final que detecta el ciclo. Los mensajes que forman parte de la cadena causal de ejecución son, por tanto, $n-(k-1)+1$. Considerando esta expresión, la latencia resultante es $(n-k+2)\cdot T$.

- Se obtiene una latencia distinta cuando todos los nodos bloqueados se disponen entre los nodos iniciadores, pero a diferencia del caso anterior, todos los nodos bloqueados son predecesores del candidato menor. A la hora de contar los mensajes que componen la cadena causal, sólo se tiene que tener en cuenta el solapamiento de los mensajes ALG de los iniciadores. En esta ordenación de nodos, el mensaje AVS resultado de la comparación de candidatos no puede adelantarse porque su generación es el efecto provocado por la recepción

de todos los mensajes ALG retransmitidos hasta alcanzar al otro candidato del ciclo. Esto implica que el número de mensajes secuenciales viene dado por $n-(k-1)+(k-1)+1$ y la latencia conseguida es $(n+1)\cdot T$.

2. $\boldsymbol{n/k \notin \mathbb{Z} \wedge k = 2}$

Como n, número de nodos del ciclo, no es divisible por k, número de iniciadores, no se podrá disponer de una distribución uniforme de los nodos bloqueados entre los iniciadores del ciclo. Las ordenaciones cuasi-balanceadas que se pueden establecer requieren que los nodos bloqueados queden repartidos como predecesores de los iniciadores de la forma más equilibrada posible, a excepción de $k-1$ nodos. Los valores de la tabla 7.7 recogen los valores de latencia para ciclos con $k = 2$ iniciadores. A la vista de estos resultados, se puede concluir que la distribución de los nodos bloqueados influye, pero en cualquier caso la complejidad temporal es $\mathcal{O}(n)$.

Distribución bloqueados	$n/k \in \mathbb{Z}$	$n/k \notin \mathbb{Z}$
balanceada/cuasi-balanceada	$((n/k)+k)\cdot T$	$(n\ div\ k+k+1)\cdot T$
predecesores del candidato mayor	$(n-k+2)\cdot T$	$(n-k+2)\cdot T$
predecesores del candidatos menor	$(n+1)\cdot T$	$(n+1)\cdot T$

Tabla 7.7: Latencia para ciclos en los que $k = 2 \wedge k < n$.

3. $\boldsymbol{n/k \in \mathbb{Z} \wedge k > 2}$

Si el número de nodos bloqueados del ciclo se puede repartir uniformemente entre los nodos iniciadores, esto es, n/k es un número entero, la longitud de la cadena causal de mensajes es diferente según la ordenación de los nodos iniciadores en el ciclo. En este apartado se va a suponer que el número de iniciadores es mayor que dos. En la sección anterior se estudiaron las configuraciones con las se obtenía el mejor y el peor coste en número de mensajes. Para analizar la latencia se va a emplear esta diferenciación.

Mejor caso:

En el llamado mejor caso de la medida de complejidad expresada en número de mensajes, los candidatos se disponen en el ciclo de mayor a menor respecto al iniciador mayor. Considerando este mejor caso, en el que no surgen mensajes *AVSRSP*, los mensajes *ALG* de los iniciadores y los que reenvían los nodos bloqueados se pueden ejecutar en paralelo. Por consiguiente, a la hora de contabilizar los mensajes *ALG* habrá que incluir el valor entero de n/k. A continuación, aparecerán tantos mensajes *AVS* como instancias haya que eliminar para quedar activa solamente la del nodo detector. Estos mensajes *AVS* resultan ser $k-1$ mensajes debido a que entre ellos no puede existir solapamiento temporal ninguno. Por último, debe añadirse a la cadena causal de mensajes, el mensaje *AVS* que permite detectar el ciclo. Resumiendo, la secuencia de mensajes consta de $n/k+(k-1)+1$ mensajes y, por tanto, la latencia supone $(n/k+k)\cdot T$.

Otras posibles localizaciones de los nodos bloqueados en el ciclo ofrecen medidas de latencia diferentes. Seguidamente, se va proceder al cálculo de tres de estas configuraciones para el mejor caso ordenación de los iniciadores.

- Nodos bloqueados entre los dos candidatos mayores del ciclo: Se solapan en el tiempo los mensajes *ALG* de los iniciadores, pero se ejecutan secuencialmente los mensajes que reenvían los nodos bloqueados. Por otra parte, la generación de los mensajes *AVS* tiene lugar al mismo tiempo que la secuencia de mensajes *ALG* que se origina entre los nodos bloqueados. Y para concluir la detecciíon del ciclo, surge un mensaje *AVS* como efecto del procesado de todos los mensajes anteriores. El número total de mensajes causales viene dado por la expresión $n-(k-1)+1$, que determina un valor de latencia de $(n-k+2)\cdot T$.

- Nodos bloqueados entre el candidato mayor y menor del ciclo: La disposición de los nodos de esta configuración impide que, salvo los mensajes *ALG* de los

iniciadores, se puedan ejecutar solapadamente los mensajes de la detección interbloqueo. Así pues, el número de mensajes de la cadena causal de esta ordenación de nodos es $n-(k-1)+(k-1)+1$ y la latencia correspondiente $(n+1)\cdot T$.

- Nodos bloqueados entre los dos candidatos menores del ciclo: En esta disposición de nodos, se puede iniciar el proceso de detección con el envío en paralelo de los mensajes *ALG* de los iniciadores. Después de esta fase, sólo será posible la comparación de dos nodos candidatos mediante un mensaje *AVS*. El resto de comparaciones entre candidatos deberá quedar aplazada hasta que los nodos bloqueados retransmitan los correspondiente mensajes *ALG*. A partir de que se procese el último de estos mensajes *ALG* redirigidos, se sucederán en el tiempo los mensajes *AVS* necesarios para reducir el número de instancias activas y el mensaje *AVS* que finaliza la detección. El valor de latencia obtenido es $n\cdot T$ y el número de mensajes que conforman la cadena causal de ejecución es $n-(k-1)+((k-1)-1)+1$.

Peor caso:

En una ordenación en la que los candidatos se sitúan de menor a mayor respecto al iniciador mayor, o lo que se ha denominado en el análisis de la complejidad en número de mensajes como peor caso, se generan, además de mensajes *ALG* y *AVS*, mensajes de tipo *AVSRSP*. Al obtener la expresión que indica el número de mensajes que componen la secuencia causal de la detección de un ciclo, se debe incluir el valor entero de n/k correspondiente a los mensajes *ALG* que no pueden solaparse en el tiempo, un mensaje *AVS* en representación de todos los mensajes *AVS* que se procesan al mismo tiempo y $k-2$ mensajes *AVSRSP* que se suceden temporalmente hasta que puede, finalmente, crearse el mensaje *AVS* de detección del interbloqueo. Sumando todos estos mensajes, resulta que la cadena de ejecución está formada por $(n/k)+1+(k-2)+1$ mensajes, o lo que es lo mismo, la medida de latencia en esta

situación es $((n/k)+k)\cdot T$. Como puede apreciarse, la latencia de una distribución uniforme de los nodos bloqueados, tanto para el mejor caso como para el peor caso de ordenación de candidatos, es la misma.

Otras posibles localizaciones de los nodos bloqueados para el peor caso de ordenación de iniciadores en el ciclo ofrecen medidas de latencia diferentes.

- Nodos bloqueados entre los dos candidatos mayores del ciclo: La diferencia fundamental de esta distribución de nodos con respecto a la equivalente del mejor caso es que todos los mensajes *AVS*, a excepción de uno, y la totalidad de los mensajes *AVSRSP* se ejecutan en paralelo con los mensajes *ALG*, que son redirigidos por los nodos bloqueados del ciclo. Esto implica que el número de mensajes causales es $n-(k-1)+1+1$, lo que proporciona un valor de latencia ligeramente superior $(n-k+3)\cdot T$.

- Nodos bloqueados entre el candidato mayor y menor del ciclo: En esta ordenación de nodos, los mensajes *ALG* de los iniciadores y los mensajes *AVS* de compación de candidatos pueden solaparse en el tiempo. Sin embargo, los mensajes *ALG* redirigidos por los nodos bloqueados, los mensajes *AVSRSP* y el mensaje *AVS* final de detección, se deben secuenciar en el tiempo porque forman una cadena causa-efecto. Por tanto, $n-(k-1)+(k-2)+1$ es la longitud de la secuencia de ejecución expresada en número de mensajes y el valor de la latencia es $n\cdot T$.

- Nodos bloqueados entre los dos candidatos menores del ciclo: La detección del ciclo en este caso requiere la ejecución secuencial de $(n-(k-1))+1+(k-2)+1$ mensajes, proporcionando así una latencia de $(n+1)\cdot T$. La generación de los mensajes *ALG* de los iniciadores y todos los mensajes *AVS*, excepto uno, que se forman para reducir el número de instancias activas pueden adelantar su ejecución porque no requieren esperar a que se originen más mensajes que los

ya procesados para su creación. En cambio, los mensajes *AVSRSP* que surgen en la detección y el mensaje *AVS* responsable de la detección final del ciclo precisan de la generación de mensajes previos para poder ser creados.

En la tabla 7.8 se resumen todos los valores de latencia encontrados en las configuraciones estudiadas. La complejidad temporal es $\mathcal{O}(n)$.

Distribución bloqueados	Mejor caso	Peor Caso
balanceada	$(n/k+k)\cdot T$	$(n/k)+k)\cdot T$
entre candidatos mayores	$(n-k+2)\cdot T$	$(n-k+3)\cdot T$
entre candidatos mayor y menor	$(n+1)\cdot T$	$n\cdot T$
entre candidatos menores	$n\cdot T$	$(n+1)\cdot T$

Tabla 7.8: Latencia en número de mensajes para ciclos en los que $n/k \in \mathbb{Z}$ y $k > 2$.

4. $\boldsymbol{(n/k)} \notin \mathbb{Z} \wedge \boldsymbol{k} > 2$

En esta situación, el número de nodos bloqueados no puede distribuirse uniformemente entre los nodos iniciadores del ciclo. La latencia de la configuraciones cuasi-balanceadas para el mejor caso y el peor caso de ordenación de los candidatos difieren en un único mensaje. En el mejor caso hay que considerar la siguiente secuencia de mensajes: $(n\ div\ k)+1$ mensajes *ALG*, $(k-1)-1$ mensajes *AVS* y el mensaje *AVS* que provoca la detección. En lo que respecta al peor caso se tiene la siguiente cadena causal de mensajes: $(n\ div\ k)+1$ mensajes *ALG*, 1 mensaje *AVS*, $k-2$ mensajes *AVSRSP* y el mensaje *AVS* final que posibilita la detección. Según estos cálculos, la latencia del mejor caso supone $((n\ div\ k)+k)\cdot T$ y la que le corresponde al peor caso, $((n\ div\ k)+k+1)\cdot T$. La tabla 7.9 muestra todos los resultados de latencia analizados.

Comparando los resultados de latencia de los distintos escenarios propuestos, se llega a la conclusión de que los valores de latencia están comprendidos entre $((n\ div\ k)+k)\cdot T$ (mínimo) y $(n+1)\cdot T$ (máximo). El límite inferior de la latencia estática corresponde a configuraciones, en las que los nodos no iniciadores se distribuyen de la

Distribución bloqueados	Mejor caso	Peor caso
cuasi-balanceada	$((n\ div\ k)+k)\cdot T$	$(n\ div\ k+k+1)\cdot T$
entre candidatos mayores	$(n-k+2)\cdot T$	$(n-k+3)\cdot T$
entre candidatos mayor y menor	$(n+1)\cdot T$	$n\cdot T$
entre candidatos menores	$n\cdot T$	$(n+1)\cdot T$

Tabla 7.9: Latencia en número de mensajes para ciclos en los que $n/k \notin \mathbb{Z}$ y $k > 2$.

forma más uniforme posible. El límite superior se registra cuando los nodos bloqueados se disponen entre el candidato mayor y menor del ciclo (mejor caso) o cuando los nodos bloqueados se sitúan entre los dos iniciadores menores del ciclo (peor caso). Cualquier otra ordenación, distinta a las recién señaladas, proporcionará un valor de latencia incluido en el intervalo indicado anteriormente. Se puede observar también que en todos los casos estáticos estudiados la latencia del algoritmo de detección de interbloqueos es lineal con respecto al número de nodos que conforman el ciclo, $\mathcal{O}(n)$.

7.3.2. Casos dinámicos

El cálculo de la complejidad temporal para evoluciones del sistema que concluyen en la formación de un ciclo entraña una doble dificultad. Por un lado, debe tenerse en cuenta que la dinámica del sistema puede ser muy variada. Las acciones de creación y borrado de esperas permiten alcanzar una configuración de nodos interbloqueados de muy diversas maneras. De ahí que, para estimar la complejidad temporal será necesario establecer unidades básicas de formación de ciclos, al igual que se ha realizado al medir el coste de comunicaciones en casos dinámicos. Por otro lado, hallar la longitud de la cadena causal de mensajes menor supone controlar la secuenciación de mensajes que admite la dinámica implementada por el algoritmo de detección y resolución de interbloqueos. En consecuencia, considerando estos dos aspectos, la metodología

adoptada para obtener medidas de latencia en sistemas dinámicos repasará las formaciones básicas de ciclo de la sección 7.2.2 para seguidamente proceder a ordenar los mensajes según su relación de causalidad.

Cadena de nodos con un iniciador que evoluciona a un ciclo con un iniciador simulado ($\boldsymbol{k} = 1$)

Los mensajes que aparecen en esta configuración son mensajes ALG redirigidos, que pasan por todos los nodos de la cadena, ya que son predecesores del nodo iniciador. La formación del ciclo implica el borrado de una serie de esperas más allá del nodo iniciador y la creación de una relación de espera que da lugar al ciclo de nodos deseado. En este proceso, se pondrá en funcionamiento el mecanismo dinámico de transmisión de identidad simulada y un nodo predecesor del iniciador hará las veces de iniciador simulado.

Si el ciclo se cerrara dejando en su interior al iniciador de la cadena, la latencia resultante sería la del caso estático equivalente, esto es, $n{\cdot}T$. Sin embargo, en el caso dinámico analizado, la complejidad temporal alcanza el valor máximo $(n{+}2m_1){\cdot}T$ que es el tiempo de propagación de todos los mensajes que aparecen en la formación y detección del ciclo. Se puede obtener una cadena causal de mensajes menor si justo después de que el iniciador lance su mensaje ALG se inicia el borrado de la espera que lo une a su predecesor que es, a su vez, el destino de su mensaje ALG. Por tanto, permitiendo el solapamiento de mensajes INF y mensajes ALG que se propagan por esperas de la cadena inicial distintas a la que ha generado el mensaje INF, resulta que la latencia obtenida es $(n{+}m_1){\cdot}T$, un valor ligeramente inferior al anterior.

Conviene recordar en este punto que los mensajes INF que surgen por el borrado de esperas se procesan secuencialmente y no se pueden tratar en paralelo. Esto es es debido fundamentalmente a que la acción del borrado de esperas está diseñada para que se asimile a la redirección de mensajes ALG que, como es sabido, es un proceso

secuencial. Tras este análisis se puede establecer que a complejidad temporaral en esta formación es $\mathcal{O}(n+m)$.

Cadena de nodos con más de un iniciador que evoluciona a un ciclo con un iniciador simulado $(\boldsymbol{k} = 1)$

A partir de los resultados obtenidos en la sección anterior, se puede ofrecer una cota máxima de latencia, para distintos números de iniciadores, que tendrá en cuenta el tiempo de propagación del total de mensajes que aparece en la formación y detección del ciclo.

- 2 iniciadores:

  $(n+2m_1+2m_2)\cdot T$ sin intercambio de mensaje entre iniciadores

  $(n+2m_1+3m_2)\cdot T$ con intercambio de mensaje entre iniciadores

- 3 iniciadores:

  $(n+2m_1+2m_2+2m_3)\cdot T$ sin intercambio de mensajes entre iniciadores

  $(n+2m_1+3m_2+3m_3)\cdot T$ con intercambio de mensajes entre iniciadores

Los valores de latencia así estimados no consideran la posibilidad de que haya mensajes que se procesen en paralelo. Del mismo modo que en el apartado anterior, se pueden obtener latencias menores si se advierte que los mensajes *INF* propios del borrado de esperas se pueden propagar al mismo tiempo que se retransmiten los mensajes *ALG* de los nodos iniciadores. Además, cabe la posibilidad de que los mensajes *ALG* de los distintos iniciadores de la cadena se envíen solapadamente en el tiempo. Los resultados que se muestran a continuación van a evidenciar que las evoluciones dinámicas que proporcionan una latencia menor son aquéllas en las que los iniciadores no han intercambiado mensajes directos de tipo *AVS* y/o *AVSRSP* antes de borrar las esperas que los unen. Dependiendo del número de iniciadores, se obtienen las siguientes medidas de latencia:

- *2 iniciadores sin intercambio de mensajes entre ellos.* Se distinguen dos situaciones diferentes según sea el número de nodos del ciclo final mayor o menor que la suma de todas las esperas rotas del proceso. Si $n \geq m_1+m_2$, la latencia viene determinada por el número de nodos del ciclo resultante y el de las esperas suprimidas entre el iniciador simulado y el iniciador real del que hereda la identidad simulada. La expresión de la latencia en esta situación es $(n+m_1)\cdot T$.

  Por otra parte, considerando que $n<m_1+m_2$, se obtiene un valor de latencia distinto en el que los mensajes *INF* que surgen en la fase dinámica de formación son determinantes para el cómputo del tiempo total consumido en la detección del interbloqueo. En este caso, la latencia se calcula como $(m_1+m_2+2)\cdot T$, donde m_1 son los mensajes *INF* que traspasan la identidad del iniciador real hasta el nodo del ciclo que hará de inicador simulado, m_2 los mensajes *INF* que transmiten la identidad del primer iniciador que se activa. La expresión contiene un sumando constante que recoge el mensaje ALG que detecta el interbloqueo de la configuración analizada y los mensajes ALG que lanzan inicialmente los iniciadores de la cadena de nodos. En el tiempo en que se propagan los mensajes ALG de los iniciadores no se puede adelantar el borrado de la espera del predecesor del iniciador, que se debe activar en primer lugar, porque se frustraría la formación dinámica que se trata de analizar.

- *2 iniciadores con intercambio de mensaje entre ellos.* Para lograr que haya una comunicación previa entre los iniciadores reales de la cadena, es preciso que el mensaje ALG del iniciador que primero se activa llegue hasta el segundo iniciador y la identidad (simulada) de este segundo iniciador sea superior a la del primero. Como ya se ha explicado con anterioridad, el resultado de esta comparación de identidades entre dos candidatos provoca la generación de un mensaje directo *AVS*. Además, la dinámica de formación del ciclo no comenzará hasta

que este mensaje *AVS* sea procesado convenientemente en su destino. El motivo por el que se impone retrasar el proceso de borrado de las esperas de los predecesores del primer iniciador activado es poder poner en marcha posteriormente el mecanismo dinámico de retroceso de mensajes *AVS* y comprobar si influye o no en la medida de la latencia.

El número de mensajes causales que se van a contabilizar en esta situación es m_1+2m_2+2. Como puede observarse, que el número de nodos del ciclo, n, sea mayor o menor que el número total de esperas rotas deja de influir en el valor de la latencia. También se aprecia que la latencia obtenida en esta situación es ligeramente superior a la del caso de dos iniciadores sin intercambiar mensajes. Esto es debido a haber obligado que el borrado de esperas se produzca tras el intercambio de mensajes de iniciadores.

Del mismo modo, hay que explicar que en el borrado de la espera que une al iniciador con su predecesor surge tanto el mensaje *INF* como el primer mensaje *AVS* en retroceso, pero no se va considerar que se propagan solapadamente. El procesado de un mensaje *AVS* se realiza si el mensaje *INF*, generado en la ejecución de la misma acción que el *AVS* en retroceso, ha sido recibido previamente. Por esta razón, para el recuento de mensajes que compone la cadena causal de ejecución menor, se ha impuesto una relación de causalidad entre ellos y, por tanto, se retarda el envío del mensaje *AVS* en retroceso. Los mensajes *AVS* en retroceso que aparecen en el tramo de esperas rotas entre iniciadores son m_2-1 porque se evita enviar un mensaje *AVS* con un sólo nodo en su ruta. Dado que la generación de estos mensajes se ha implementado en el algoritmo como un mecanismo secuencial, su propagación queda totalmente solapada con el envío secuencial de los mensajes *INF* del mismo tramo de esperas rotas de la cadena de nodos. En consecuencia, el valor de la latencia no se verá incrementado por

la aparición de mensajes *AVS* en retroceso.

A continuación, la tabla 7.10 recoge todas las medidas realizadas para el caso en el que el ciclo final sólo hay un iniciador simulado ($k = 1$). Si se considera que en la configuración inicial había 2 iniciadores, i_1 e i_2, siendo i_2 el iniciador más alejado del iniciador simulado, se obtienen diferentes valores de acuerdo a la relación que hay entre el número de nodos del ciclo, n, y la suma de las esperas rotas (columna 2 de la tabla). Además de estudiar todas esos supuestos, se ha calculado la latencia de acuerdo a la relación de las identidades de los iniciadores y, por consiguiente, al posible intercambio de información entre ellos antes del borrado de esperas de la cadena de nodos inicial.

Número de iniciadores iniciales activados	Relación	Latencia
1		$(n+m_1)\cdot T$
2	$n \geq m_1+m_2$	$(n+m_1)\cdot T$ $(i_1>i_2)$
	$n<m_1+m_2$	$(m_1+m_2+2)\cdot T$ $(i_1>i_2)$
		sin intercambiar información
	$n \geq m_1+m_2$	$(n+m_1)\cdot T$ $(i_1<i_2)$
	$n<m_1+m_2$	$(m_1+m_2+2)\cdot T$ $(i_1<i_2)$
		sin intercambiar información
	$n \geq m_1+2m_2$	$(n+m_1)\cdot T$ $(i_1>i_2)$
	$n<m_1+2m_2$	$(m_1+2m_2+2)\cdot T$ $(i_1>i_2)$
		intercambiando información
	$n \geq m_1+2m_2$	$(n+m_1)\cdot T$ $(i_1<i_2)$
	$n<m_1+2m_2$	$(m_1+2m_2+1)\cdot T$ $(i_1<i_2)$
		intercambiando información

Tabla 7.10: Latencia de ciclos con un iniciador simulado ($k = 1$)

Cadena de nodos que evoluciona a un ciclo con un iniciador simulado y otro real ($\boldsymbol{k} = 2$)

Las configuraciones que se van a analizar seguidamente evolucionan de tal forma que el ciclo presenta dos iniciadores. Uno de ellos no ha modificado sus relaciones de espera con los nodos que le preceden en la cadena inicial (iniciador real, i_2) y el otro se considera un iniciador simulado (i_1), porque, ha experimentando ciertos cambios con respecto a la configuración inicial al activarse. La activación de este nodo hace que tenga que adquirir una identidad simulada que le haga comportarse como el nodo iniciador que ya no tiene presencia final en el ciclo. Si el estado del nodo antes de convertirse en iniciador simulado era *dummy*, además de adquirir una identidad simulada también tiene que cambiar a estado *candidate*. En el caso de ser *candidate* con *inf_need* a *true*, mantiene el estado pasando *inf_need* a *false* y adquiere una nueva identidad simulada. Es obvio que, al igual que en los casos estáticos, según la distribución de los nodos bloqueados entre los iniciadores finales del ciclo, se obtengan valores de latencia diferentes.

1. *Estado previo del iniciador simulado: dummy.*

La tabla 7.11 muestra un resumen del número de mensajes causales necesario para detectar un interbloqueo. Se ha elegido el mejor y el peor caso encontrado, al variar la distribución de los nodos bloqueados que hay entre los iniciadores. Asimismo, se proporcionan los registros obtenidos cuando el detector del ciclo es el iniciador simulado ($i_1 > i_2$) y cuando el detector es el iniciador real ($i_1 < i_2$). Se observa que las detecciones del iniciador real precisan de más mensajes porque se pone en marcha el retroceso de mensajes *AVS* y no se puede solapar totalmente con los mensajes *INF* que transmiten la identidad simulada que adopta el iniciador simulado.

La misma tabla recoge las medidas obtenidas cuando la configuración inicial cuenta con tres iniciadores, pero dos de ellos se activan. En esta situación, el ciclo final

Número de iniciadores iniciales activados	Relación/ Detector	Latencia
1	$i_1<i_2$	$((n+k)+m_1+3)\cdot T$
	iniciador real	$((n+k)+m_1+1)\cdot T$
	$i_1<i_2$	$((n+k)+m_1+1)\cdot T$
	iniciador simulado	$((n+k)+m_1)\cdot T$
2	$i_1>i_2>i_3$	$((n+k)+2m_1+m_2-1)\cdot T$
	iniciador simulado	$((n+k)+2m_1+m_2-3)\cdot T$
	$i_1>i_3>i_2$	$((n+k)+3m_1+2m_2+1)\cdot T$
	iniciador simulado	$((n+k)+3m_1+m_2-4)\cdot T$
	$i_2>i_1>i_3$	$((n+k)+3m_1+m_2-1)\cdot T$
	iniciador simulado	$((n+k)+2m_1+m_2-4)\cdot T$
	$i_2>i_3>i_1$	$((n+k)+3m_1+m_2-4)\cdot T$
	iniciador simulado	$((n+k)+2m_1+m_2-4)\cdot T$
	$i_3>i_1>i_2$	$((n+k)+2m_1+m_2+1)\cdot T$
	iniciador real	$((n+k)+2m_1+m_2-4)\cdot T$
	$i_3>i_2>i_1$	$((n+k)+2m_1+m_2)\cdot T$
	iniciador real	$(n+k)+2m_1+m_2-3)\cdot T$

Tabla 7.11: Latencia, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*dummy* previamente) y un iniciador real ($k = 2$).

también dispone de dos iniciadores ($k = 2$), uno real i_3 y otro simulado que adopta la identidad simulada de uno de los iniciadores activados, i_1 ó i_2. Los valores de latencia que se han anotado son muy variados porque hay una gran diversidad de configuraciones. Dado que, antes de empezar a borrar las esperas que conducen a la formación del ciclo, se intercambian todos los mensajes posibles entre iniciadores, se contabilizan cantidades de mensajes diferentes, dependiendo de si la identidad simulada procede del iniciador activado más alejado o del más cercano. También influye qué mecanismos dinámicos se necesitan para lograr la detección del interbloqueo. Por

Número de iniciadores iniciales activados	Detector	Latencia
1	iniciador real	$((n+k)+2m_1-3)\cdot T$ ...
	iniciador simulado	$((n+k)+2m_1-6)\cdot T$ ...
2	iniciador real	$((n+k)+2m_1+m_2-4)\cdot T$ ...
	iniciador simulado	$((n+k)+2m_1+m_2-3)\cdot T$ ...

Tabla 7.12: Latencia, según el número de iniciadores iniciales activados, para ciclos en los que hay un iniciador simulado (*candidate* previamente) y un iniciador real ($k = 2$).

ejemplo, la conversión de mensajes almacenados en *set_st_avs* en mensajes *AVSRSP* permite que la detección del interbloqueo requiera menos mensajes porque su información está actualizada. Sin embargo, la latencia se incrementa notablemente cuando el iniciador simulado detecta el ciclo a través de un mensaje *AVS* en retroceso y no de forma directa porque las rutas contenía información obsoleta.

Considerando todos las situaciones planteadas y a la vista de los datos anotados en la tabla 7.11, se observa que la complejidad temporal es $\mathcal{O}(n + m)$, donde m representa las esperas rotas entre iniciadores iniciales activados y el simulado.

2. *Estado previo del iniciador simulado: candidate* con *inf_need = true*.

En el siguiente análisis, el número de iniciadores del ciclo final es ($k = 2$), pero el iniciador simulado antes de activarse ya era *candidate* y había intercambiado mensajes con otros candidatos de la cadena inicial (*inf_need = true*). Esta condición impone que las cadenas de nodos en el inicio tengan al menos tres nodos iniciadores. Al igual que en el caso precedente, se han estudiado diferentes configuraciones. Dependiendo

de la disposición de los iniciadores en la cadena inicial, de la distribución de los nodos bloqueados en el ciclo final y del número de esparas que se rompen en la evolución considerada, se contabiliza un número distinto de mensajes causales. Los mecanismos dinámicos que afectan en cualquiera de los casos planteados son los que han sido diseñados para que se ejecuten secuencialmente: el retroceso de mensajes *AVS* cuando finaliza en una detección y la propagación de los mesnajes *INF*.

Observando los valores de la tabla 7.12, vuelve a extraerse la misma conclusión respecto a la latencia. La complejidad temporal medida en número de mensajes causales para la evolución dinámica considerada es $\mathcal{O}(n + m)$. Este resultado se puede generalizar a configuraciones más complejas porque éstas se pueden obtener como combinación de las configuraciones básicas ya estudiadas.

7.4. Algunos aspectos sobre la implantación real del algoritmo

A la hora de diseñar un algoritmo es muy habitual asumir un modelo idealizado del sistema que facilite la descripción y posterior demostración formal del algoritmo. Por esta razón, en la implantación de un algoritmo en un sistema real pueden surgir problemas en la ejecución o comportamientos no deseados del algoritmo, que eran imposible prever en un entorno ideal. Contrariamente, también es posible que la implantación de un algoritmo en un sistema concreto permita optimizar algunas de sus prestaciones. La pérdida de cierto grado de la generalidad impuesta en el diseño de un algoritmo puede mejorar el rendimiento del mismo porque su complejidad espacial disminuya

7.4.1. Abortos espontáneos

Un algoritmo que verifique la condición de seguridad supone la ausencia de abortos espontáneos. Por eso, se suele suponer que el sistema estará libre de abortos espontáneos. Sin embargo, en un sistema real no se puede garantizar que el algoritmo no detecte y resuelva falsos interbloqueos causados por abortos espontáneos.

Con objeto de reducir el número de esos falsos interbloqueos, los algoritmos con memoria podría incorporar un esquema que lanzara, tras el proceso de detección, una acción para que abortara el nodo detector y además se eliminaran mensajes irrelevantes para el devenir del sistema. En cierta medida, la acción $Abort_i$ del algoritmo planteado en esta tesis, incluye mecanismos para recuperar la información relevante y descartar la irrelevante. De esta forma, aunque no se asegura que no haya falsas aborciones, se reduce su número.

7.4.2. Condiciones de disparo

En este algoritmo no se establece el momento en que un cierto nodo inicia la instancia del algoritmo. En la iniciación sólo se exige que el nodo tenga predecesores y esté bloqueado con una identidad simulada conocida. Cumpliéndose estas condiciones el nodo tiene habilitada la acción $initiate_i$, estando en disposición de mandar un mensaje ALG cuando se ejecute la acción. Como se ha comentado en los apartados anteriores, el número de mensajes que emplea el algoritmo puede llegar a ser muy grande y está directamente relacionado con el número de instancias que se inicien en el sistema. Con el fin de reducir las iniciaciones del algoritmo en una implementación real del algoritmo, se deberían adoptar una serie de medidas que permitieran la optimización del número de iniciaciones del algoritmo. Es bien sabido que no siempre que se inicia el algoritmo existe un interbloqueo en el sistema. Por eso, no es necesario provocar de manera continuada la iniciación de instancias del algoritmo, ya que puede

que haya un bloqueo simple que se elimina tras un corto espacio de tiempo.

Se podría pensar en un mecanismo de *time-out* para que cada nodo iniciaría sus instancias transcurrido un tiempo desde que se recibió la notificación de la existencia de la espera. El periodo de *time-out* se podría decidir en función de diversos parámetros como la prioridad del nodo. Incorporando esta temporización de las instancias, se consigue que todos los nodos creen sus instancias del algoritmo en un tiempo finito y controlable. En algunos casos se podría además forzar a que se creen las instancias de nodos de alta prioridad en primer lugar. En resumen, con el ajuste de las condiciones de disparo cabe la posibilidad de reducir el número de mensajes que emplea el algoritmo en un sistema real.

7.4.3. Optimización del algoritmo

Los mecanismos de tratamiento de los mensajes con información que no es correcta o es irrelevante es un aspecto de vital importancia para el correcto funcionamiento del algoritmo. Por eso, tanto en los mensajes *ALG* como *AVSRSP* se valida la característica temporal. También se impide que la detección se desarrolle a no ser que todos los nodos implicados conozcan su identidad, *status.id* vale *known*. Mientras *status.id* tenga el valor *unknown*, el procesado de los mensajes quedará suspendido temporalmente. Además, la recepción y tratamiento de mensajes está restringida a ciertos valores posibles de la variable *status.alg*. La variable *t_unk* también sirve para paralizar en cierta medida el proceso de detección hasta acomodar la información existente a la activación de nodos, que transmitieron información fundamental para detectar un ciclo. Del mismo modo, la variable tipo conjunto *set_st_avs* también guarda información usada en la detección, que puede ser reutilizada si desaparecen esperas registradas en la ruta de sus mensajes almacenados. Incluir estas variables no afecta a la demostración de seguridad ni al análisis de complejidad estática porque

son mecanismos que se ponen en funcionamiento en evoluciones dinámicas del sistema. La adaptación y mejora del comportamiento en entornos cambiantes tiene como contrapartida el aumento de la memoria necesaria para implantar el algoritmo.

Los nodos abortan en cuanto reciben información que evidencia la existencia de un ciclo. Es posible que en el mensaje *ALG* haya información referente a una ruta cuya parte final no es correcta, pero se cuenta con unas reglas que lo validan igualmente. Lo mismo sucede con los mensajes *AVS*: el nodo detector puede aparecer con un tiempo obsoleto, pero los criterios de detección del ciclo permiten obviar esa situación. Mantener informaciones en las variables que permanecen en el sistema tras el aborto de un nodo, aunque ya no sean ciertas en su totalidad, va a hacer posible que la reinicialización del algoritmo por parte de alguno de los nodos del ciclo extinguido no requiera poner en funcionamiento el proceso de detección desde un principio. Por otra parte, eso permite que el algoritmo tenga una complejidad menor en número de mensajes y menor en tiempo de latencia para evoluciones dinámicas del sistema. La condición de progreso en este tipo de reinicializaciones se verifica porque el número de procesos del sistema es finito en cualquier instante y existe una relación de orden establecida que no se altera tras el aborto de un nodo. Siempre habrá un candidato con una identidad simulada única mayor a la del resto de candidatos del sistema.

Seguidamente, se describen dos modalidades de gestor del sistema a los que puede adaptarse el algoritmo para optimizar su rendimiento.

1. *Gestor de sistema que implementa un esquema 2PL:* El bloqueo en dos fases es típico en los gestores de sistemas de bases de datos. Los procesos solicitan todos los recurso que necesitan al principio de su computación. Después de liberar un recurso, el proceso no puede hacer nuevas peticiones. De acuerdo con estas condiciones, las esperas no pueden aparecer en el sistema más que una vez. Este hecho se reflejaría en el algoritmo eliminado la variable del tiempo de activación que distingue las distintas versiones de una misma espera. Las

comprobaciones temporales en la recepción de mensajes se suprimirían. Como consecuencia de estas simplificaciones, se reduce el espacio de memoria necesario para el funcionamiento del algoritmo.

2. *Uso de vectores de tiempo:* Al diseñar algoritmos distribuidos se hace patente la imposibilidad de conocer el estado global del sistema en un momento concreto. Para subsanar en cierta medida esa incapacidad, se recurre a descubrir cuál es el ordenamiento causal de los eventos que conoce un determinado proceso. Para establecer este ordenamiento se hace uso de generalizaciones de los relojes temporales de Lamport [50]. Una de las generalizaciones más habitual en los trabajos de interbloqueo distribuido es la de Fidge [125]. En ella, los procesos del sistema tienen asociado un vector de tiempo y un reloj lógico. Cada proceso incrementa su entrada del vector de tiempo al ejecutar un evento y actualiza el resto de entradas de su vector al recibir los vectores de otros procesos a través de mensajes. En el algoritmo de resolución de interbloqueos que se ha presentado, no se debería modificar el reloj lógico de los procesos cuando éstos están bloqueados y se realizan eventos relacionados con la resolución. Esta información temporal puede permitir al gestor del sistema avanzar actuaciones (envío de mensajes o modificación de esperas) propias de los escenarios dinámicos. Este mecanismo reduciría el número de mensajes de información y, en consecuencia, el número de reinicializaciones en el algoritmo.

7.5. Análisis de los resultados

Si se comparan las medidas de complejidad obtenidas para el algoritmo con las de algoritmos previos, se puede observar una notable reducción del orden de de la complejidad en número de mensajes. Esta reducción se consigue sin aumentar el orden de la complejidad temporal. Como contrapartida, el tamaño de los mensajes

Algoritmo	Número de mensajes	Latencia
Mitchell	$\mathcal{O}(n^2)$	n
Sinha	$\mathcal{O}(n^2)$	$2n-1$
Choudhary	$\mathcal{O}(n^2)$	$2n-1$
Kshemkalyani	$\mathcal{O}(n^2)$	$2n-1$
Glez. de Mendivil	$\mathcal{O}(n^2)$	$2n-2$
Kim	$\mathcal{O}(n^2)$	n
Prieto	$\mathcal{O}(n)$ $2n-1$	$n/2+1$
Castillo escenarios estáticos	$\mathcal{O}(n)$ $n+2k-2$(peor caso)	$n+1$ (peor caso)

Tabla 7.13: Comparación de complejidades de diversos algoritmos en esceanrios estáticos.

que emplea el algoritmo es variable, pudiendo alcanzar una longitud considerable en interbloqueos formados por un gran número de nodos. Sin embargo, este inconveniente se transforma en una ventaja al comprobar que, gracias a la información que se recoge de los nodos del ciclo, el algoritmo permite detectar y resolver interbloqueos con coste lineal siguiendo la evolución dinámica del sistema. En las propuestas anteriores está funcionalidad no era posible y ante cualquier cambio en el grafo de esperas del sistema se debían ejecutar nuevas instancias del algoritmo.

En la tabla 7.13 se pueden observar las complejidades de diversos algoritmos. De entre los algoritmos cuadráticos en número de mensajes se puede destacar tanto el de Kshemkalyani [56] como el de Glez. de Mendivil [73]. La diferencia fundamental de ambos es que el primer algoritmo emplea memoria para la resolución de interbloqueos y el segundo no requiere de memoria para solucionar el problema del interbloqueo. El número de mensajes en ambos es proporcional a n^2, pero cada modificación en las esperas del sistema hace que surjan n mensajes por espera en el primero y $(n+1)^2$

Algoritmo	Número de mensajes	Latencia
Prieto	$\mathcal{O}(k{\cdot}(n{+}m))$	—
Castillo	$\mathcal{O}(n{+}m)$	$\mathcal{O}(n{+}m)$
escenarios dinámicos	$(n{+}2k{-}2){+}m$	$n{+}1{+}m$

Tabla 7.14: Comparación de complejidades de algoritmos lineales en escenarios dinámicos.

mensajes en el segundo. Esto implica que la complejidad de los algoritmos se incrementaría ante cambios en las relaciones de espera alcanzando complejidades $\mathcal{O}(n^2)$ y $\mathcal{O}(n^3)$ para el algoritmo con memoria y sin memoria, respectivamente.

Los dos últimos registros de la tabla 7.13 corresponden a algoritmos que alcanzan complejidades lineales en número de mensajes. El algoritmo de Prieto [112] (sin memoria) resuelve interbloqueos con un número de mensajes proporcional al número de nodos que forman el interbloqueo. En su caso, la modificación de las esperas incrementaría su coste notablemente, porque precisaría de $n{+}1$ mensajes nuevos por cada modificación del interbloqueo considerado (ver tabla 7.14). A diferencia del algoritmo de Prieto, el que se propone en esta tesis cuenta con memoria. Sin alterar la complejidad en número de mensajes (sigue siendo lineal, $n{+}2k{-}2$ en el peor caso estático), el algoritmo está dotado de mecanismos que le hacen adaptarse a evoluciones dinámicas del sistema. Esto permite que la complejidad en número de mensajes sea $(n{+}2k{-}2){+}m$ en el peor caso, donde m es el número de esperas rotas en la formación del ciclo y k el número de iniciadores del ciclo final.

Es obvio que la latencia del algoritmo propuesto se ve penalizada por la inclusión de la memoria fundamental para abordar la característica dinámica del mismo. En los casos estáticos, cuando el número de nodos del interbloqueo es equivalente al de iniciadores, la latencia aun siendo lineal es superior a la que proporciona el algoritmo de Prieto. Sin embargo, la latencia en casos dinámicos mantendría su linealidad y la del algoritmo de Prieto la abandonaría. En la tabla 7.14 no se proporciona ningún

valor de latencia para el algoritmo de Prieto. Aunque se podría aventurar que, en el mejor de los casos, la medida temporal alcanzará valores cuadráticos, no se puede estimar porque en su trabajo no se describe ningún método para la elimnación de información residual. Los mensajes que deberían surgir para evitar inconsistencias ante un cambio en las esperas del sistema no están considerados y es evidente que influyen directamente en el orden causal de los mensajes necesarios para la resolución de un interbloqueo.

Como complemento a estos resultados, se podría añadir la medida de latencia del mejor caso posible. Si la latencia del algoritmo se mide desde el momento que se crea el ciclo hasta que se rompe, el algoritmo ha podido adelantar el envío de la mayor parte de mensajes. Suponiendo que en un determinado instante existen todos los arcos del ciclo excepto uno y que en un instante posterior tiene lugar el bloqueo del nodo que cierra el ciclo, la tarea de contar el número de mensajes necesarios para detectar el interbloqueo se reduce notablemente. Por ejemplo, en un escenario estático con sólo un iniciador, podría bastar con dos mensajes para que el algoritmo cumpla con su cometido. Por lo tanto, se puede asegurar que el mejor caso del algoritmo en tiempo de latencia es $\mathcal{O}(1)$. Ni siquiera en otros algoritmos sin memoria se consigue este valor de complejidad temporal (en el algoritmo de Prieto la medida de latencia en el mejor caso supone $\mathcal{O}(n)$ mensajes).

Capítulo 8

Conclusiones y principales aportaciones

En este trabajo se propone una solución al problema del interbloqueo distribuido para sistemas con modelo *Single Request*, *SR*, de petición de recursos. El principal objetivo que se planteó al inicio del trabajo fue desarrollar un algoritmo que cumpliera, con el menor coste posible, los criterios clásicos de corrección. Los capítulos 6 y 7 de esta memoria muestran que se ha logrado dicho objetivo. El algoritmo presentado es eficaz desde dos puntos de vista: todos los interbloqueos que aparecen en el sistema se resuelven en un tiempo finito y nunca se abortan procesos que no estén interbloqueados realmente. Es importante volver a señalar que todo ello se consigue minimizando la cantidad de mensajes necesarios.

A lo largo del trabajo se han tratado de justificar formalmente todos los resultados. Por este motivo, tanto la descripción del funcionamiento del medio y del algoritmo, como la especificación del problema y la demostración de corrección se ha proporcionado en términos del modelo formal de Autómatas de Entrada/Salida. Este modelo facilita la comprensión del algoritmo y la demostración de su corrección.

De manera previa a la fase del diseño del algoritmo, se ha establecido una analogía entre los problemas de elección de líder y de detección de interbloqueos (capítulo 2). Las soluciones al problema de interbloqueo que se centran únicamente en su detección

suelen ser muy similares a las soluciones la problema de elección de líder, dado que se limitan a señalar a un candidato para resolver el interbloqueo. Dicho de otro modo, al diferenciar un nodo entre los interbloqueados, se está conjugando el mecanismo de elección de líder con otro mecanismo que asegura la existencia de un ciclo. Esta relación resulta obvia si se observan las similitudes del algoritmo de elección de líder de LeLann [97] y los algoritmos de interbloqueo de tipo *edge-chasing*. También es evidente que una de las similitudes es su carácter cuadrático en número de mensajes. Este hecho llevó a concluir que se podría extender esta similitud tal caso de un algoritmo de elección de líder con coste lineal. El algoritmo de detección y resolución de interbloqueos que se ha construido imita, en su funcionamiento estático, a la elección de líder para redes completas de [113]. La elección de este algoritmo viene motivada por dos razones: su linealidad y la topología de red que asume, la cual evita muchos de los mensajes que requieren los algoritmos *edge-chasing* habituales.

Con el fin de adaptar del algoritmo se ha seguido un proceso que podría ser generalizable a otros problemas dinámicos que tengan relación con problemas estáticos. En primer lugar, se trató de diferenciar qué partes de la solución estática que se ven comprometidas por la introducción del dinamismo. En segundo lugar, se analizaron los motivos por los que el dinamismo hacía fallar a la solución estática. Por último, se integraron en el algoritmo elementos que corrigieran los problemas. En el algoritmo estudiado en esta tesis, los problemas que introducía el dinamismo eran fundamentalmente dos: la posibilidad de que el iniciador de una computación la abandonase y la modificación de los datos del sistema que el algoritmo ya había recogido. El primer problema se ha solucionado introduciendo un mecanismo que convierta a uno de los procesos que siguen en la computación en representante del iniciador que la abandona. El segundo ha precisado forzar a que los cambios en un dato sean comunicados a aquellos nodos que conococían el dato modificado. Cabe destacar que esta segunda solución obliga a incorporar mecanismos que ordenen la ejecución de algunas acciones

y a incluir mucha más información en los mensajes.

Otra de las contribuciones más destacadas del presente trabajo es la complejidad lineal del algoritmo. La mayor parte de los algoritmos de resolución de interbloqueos para el modelo *Single Request* aseguran que cuentan con complejidades en número de mensajes de $\mathcal{O}(n^2)$, donde n es el número de nodos que conforman el interbloqueo. El algoritmo de esta tesis sólo requiere $n+2k-2$ mensajes, $\mathcal{O}(n)$, para resolver un interbloqueo en el que intervienen n nodos de los que k son iniciadores del algoritmo. Podría argumentarse que el algoritmo de Prieto publicado en [112] ya proporcionaba un coste lineal ($2n-1$ mensajes). Sin embargo, hay que recordar que los estudios de complejidad de los trabajos previos se centran en escenarios estáticos. Muchos autores han indicado que la mayoría de los algoritmos conocidos presentan una complejidad $\mathcal{O}(n^3)$ en escenarios dinámicos. Si se analiza el algoritmo de Prieto en escenarios cambiantes, se aprecia éste deja de ser lineal. Durante la formación del interbloqueo pueden haber existido n instancias que hayan transmitido $\mathcal{O}(n)$ mensajes cada una. En el algoritmo de esta tesis la coordinación entre diferente instancias del algoritmo, así como la inclusión de elementos de memoria, permite valorar la información útil y desechar la información incorrecta, no sólo en escenarios estáticos, sino también en escenarios dinámicos. La exploración eficiente de las relaciones de espera que surgen en el sistema hace que los nodos, al ejecutar la copia del algoritmo, utilicen información ya recopilada por otros nodos. En consecuencia, la colaboración estrecha entre los nodos del sistema hace que no se repitan tareas que ya se habían realizado en momentos anteriores a la formación de un interbloqueo.

Para lograr el carácter dinámico del algoritmo sin afectar a la linealidad del coste en número de mensajes, se han incorporado novedosos mecanismos como, por ejemplo, el uso y propagación de una identidad simulada validada durante todo el proceso de creación de un interbloqueo. Los mecanismos que requiere la adaptación a escenarios dinámicos influyen en la complejidad temporal, porque se necesita retardar

la recepción de algunos mensajes para controlar los cambios en el sistema. Debido a esto, la ordenación causal de los mensajes se convierte en un factor determinante en el cálculo de la medida de latencia. A pesar de ese inconveniente, la complejidad temporal conseguida también es lineal.

Considerando todo lo expuesto, se puede concluir que el estudio de complejidad del algoritmo es en sí mismo una aportación adicional. No se ha encontrado en la literatura otras propuestas que analicen el coste de los algoritmos incluyendo no sólo los mensajes transmitidos durante la fase de detección/resolución del interbloqueo, sino también todos aquéllos que pudieran haber sido enviados durante su fase de formación. Este hecho lleva a un problema que hay que mencionar. No es posible realizar una comparación objetiva de la complejidad dinámica del algoritmo con la de otros algoritmos. En primer lugar por falta de soluciones al problema que sean lineales a excepción del algoritmo de Prieto ya citado anteriormente. Este algoritmo, con el que sería factible la comparación, sólo ofrece cálculos de complejidad en situaciones estáticas en las que no hay interrelación entre una resolución y posteriores, y/o evoluciones diversas antes de una resolución. Además, la ausencia de cualquier tipo de tratamiento para las situaciones dinámicas y su modo de funcionamiento en las estáticas indican que la ejecución de instancias del algoritmo por nodo se dispara alcanzando complejidades cuadráticas.

8.1. Líneas futuras de trabajo

El trabajo reflejado en esta memoria comprende fundamentalmente el diseño, la prueba de corrección y el análisis de las prestaciones del algoritmo, pero al mismo tiempo deja entrever nuevas ideas u horizontes de trabajo:

- Completar el estudio de la complejidad mediante un análisis estadístico en el que se simule el comportamiento del algoritmo en los más diversos casos y

considerando diferentes modelos de sistema. Sólo elegir los parámetros usados para modelar los sistemas entrañaría una gran dificultad porque su influencia en el funcionamiento de los algoritmos no es evidente. Según los valores de los parámetros podría haber grandes diferencias de funcionamiento, comportamientos similares o incluso algún comportamiento de interés. Estos modelos estadísticos, junto con un análisis de la carga del sistema y del tamaño medio de los interbloqueos, ayudaría a elegir qué algoritmo de los existentes implantar en un sistema real concreto. Otra posibilidad sería la creación de un banco de pruebas que permitiese la comparación objetiva de diferentes propuestas.

- Emplear en otros problemas de interés la idea de diseño que ha servido para reducir la complejidad (en número de mensajes) en el problema de la detección y resolución de interbloqueos. El algoritmo de elección de líder de [111] y [113] ha inspirado el algoritmo de [112] y el presentado en esta tesis, respectivamente. Este hecho avala la posibilidad de establecer analogías entre diferentes problemas y aprovechar soluciones de problemas. Es probable que las técnicas que han permitido dinamizar el algoritmo de elección de líder puedan utilizarse para obtener soluciones en sistemas dinámicos a algunos problemas ya resueltos en sistemas estáticos.

- Adaptar el algoritmo a otros entornos. Como se ha explicado, el interbloqueo puede aparecer en muy diferentes ámbitos de aplicación. Cada uno de ellos puede tener características que permitan simplificar algunas partes del algoritmo. Esta línea de trabajo ya se ha emprendido. Se ha hecho una primera adaptación del algoritmo para resolver el problema de interbloqueo en aplicaciones de procesado de la señal y multimedia (*streaming* de datos). Los ajustes realizados para este tipo de aplicación originan una nueva versión del algoritmo que puede consultarse en [126] y [127].

Apéndice A

Modelo de autómatas de Entrada/Salida

El modelo de autómatas de Entrada/Salida fue desarrollado por Lynch y Tuttle [114], [128]. En [54] se analizan los motivos para su introducción como modelo formal en el tratamiento de sistemas concurrentes y distribuidos. En ese mismo trabajo se pueden encontrar también varios ejemplos de aplicación del modelo.

Este modelo permite la realización de demostraciones jerárquicas de algoritmos, empleando una serie de objetos (autómatas, módulos de ejecución, módulos de planes, etc.) y de una serie de operadores que relacionan los diferentes objetos. El modelo inicial de los trabajos [128] y [54] ha sido mejorado a lo largo del tiempo y en posteriores versiones se incluyen características que posibilitan la representación de propiedades en tiempo real [129], cálculos estadísticos [130], etc.

En este apéndice se presenta un resumen del modelo de autómatas de Entrada/Salida, basado en los trabajos de Tuttle [114] y Fekete [117] y de los resultados necesarios para la realización del trabajo de esta tesis.

En primer lugar se van a describir los componentes de un autómata y algunos de los operadores que incluye el modelo de autómatas de Entrada/Salida. A continuación, se

presentarán otros objetos del modelo como son los módulos de ejecución, los módulos de planes y las especificaciones. Finalmente, se enunciarán algunas propiedades y teoremas que resultan de gran utilidad cuando se lleva a cabo una prueba de corrección jerárquica.

A.1. Definición de un autómata de Entrada/Salida

El principal objeto del modelo de autómatas de Entrada/Salida es una máquina de estados denominada autómata de Entrada/Salida o, simplemente, *autómata*. Antes de escribir una definición completa de autómata es necesario introducir el concepto de *signatura de acciones*.

Mediante una signatura de acciones se pueden identificar las interacciones de un algoritmo con el entorno donde se ejecuta y clasificar dichas interacciones según sea su origen y su medio de actuación. En consecuencia, una signatura de acciones, S, está compuesta por tres conjuntos de acciones, esto es, $S = \{in(S), out(S), int(S)\}$, donde $in(S)$ es el conjunto de acciones de entrada, $out(S)$ el conjunto de acciones de salida e $int(S)$ el conjunto de acciones internas. Como no existen acciones que pertenezcan a más de uno de esos conjuntos, estos tres conjuntos de acciones son disjuntos.

Un algoritmo, o los objetos que lo modelan, reciben acciones de entrada que proceden del entorno y que pueden desencadenar otras acciones internas y/o de salida. A su vez, los objetos pueden ejecutar acciones de salida cuyo objetivo será modificar las características del entorno. Por último, las acciones internas que generan los propios objetos del algoritmo permiten manipular los objetos del algoritmo sin cambiar las propiedades del medio, es decir, sus efectos repercuten sólo en el objeto que las controla u otros objetos del algoritmo.

En algunas ocasiones resulta conveniente agrupar las acciones que componen la signatura de una forma diferente. En esa clasificación alternativa se cuenta con los

siguientes conjuntos de acciones de S: el *conjunto de acciones externas*, el *conjunto de acciones localmente controlables* y el *conjunto (total) de acciones*. El conjunto de acciones externas de S, $ext(S)$, está formado por las acciones de entrada y de salida, $ext(S) = in(S) \cup out(S)$. El conjunto de acciones localmente controlables de S, $local(S)$, lo forman las acciones internas y de salida, $local(S) = int(S) \cup out(S)$. Por último, el conjunto de acciones de S, $acts(S)$, se define como la unión de los tres conjuntos de acciones, o sea, $acts(S) = in(S) \cup out(S) \cup int(S)$.

Un autómata A consta de los siguientes elementos:

- un conjunto de estados de A, $states(A)$,
- un conjunto no vacío de estados iniciales, $start(A)$, tal que $start(A) \subseteq states(A)$,
- una signatura de acciones de A, $sig(A)$,
- una relación de transición de A, $steps(A)$, tal que $steps(A) \subseteq states(A) \times acts(A) \times states(A)$, con la propiedad de que para todo estado $s \in states(A)$ y para toda acción de entrada $\pi \in in(A)$ hay una transición $(s, \pi, s\prime)$ en $steps(A)$,
- una relación de equivalencia que hace corresponder a una partición, $part(A)$, del conjunto $local(A)$ con un número contable de clases (de equivalencia).

Para referirse a los conjuntos de acciones de un autómata $in(sig(A))$, $out(sig(A)$,... se va a emplear la notación simplificada $in(A)$, $out(A)$,...

Los pasos de un autómata A se hacen corresponder con los elementos de $steps(A)$. Así pues, cada elemento de $steps(A)$ es un posible paso de la computación del sistema que representa el autómata. Se dice que π está habilitada en el estado s, si $(s, \pi, s\prime)$ es un paso del autómata.

Se llama *fragmento de ejecución* de A a una secuencia finita $s_0\ \pi_1\ s_1 \ldots s_{k-1}\ \pi_k\ s_k$ o infinita $s_0\ \pi_1\ s_1 \ldots s_{k-1}\ \pi_k\ s_k \ldots$ que alterna estados y acciones de A tal que para

todo i, $(s_i, \pi_{i+1}, s_{i+1})$ es un paso de A. Todo fragmento de ejecución que comienza en un estado inicial se denomina *ejecución*. Se dice que un estado s de A es *alcanzable* si dicho estado aparece en alguna ejecución de A. El conjunto de todas las posibles ejecuciones de un autómata se denota $execs(A)$.

Una *ejecución equitativa* de un autómata A es una ejecución $\alpha \in execs(A)$ tal que, para toda clase C de $part(A)$, se cumple que si α es finita, entonces no hay ninguna acción de C habilitada en el último estado de α, mientras que si α es infinita, contiene infinitamente a menudo acciones de C, o bien estados en los que ninguna acción está habilitada.[1] El conjunto de las ejecuciones equitativas de A se denota $fairexecs(A)$.

La partición de un autómata, $part(A)$, se emplea para establecer las ejecuciones equitativas, de manera que queden identificados los componentes del sistema que es modelado por el autómata. Cada clase de la partición representa un conjunto de acciones de un componente determinado del sistema.

Otro concepto útil es el operador proyección. Si β es cualquier secuencia de elementos de algún alfabeto y Φ es un conjunto de dichos elementos, $\beta \mid \Phi$ representa la subsecuencia de β que contiene sólo las ocurrencias de los elementos de Φ. Al aplicar este operador, se obtiene lo que se conoce como *plan* de una ejecución α, esto es, $sched(\alpha) = \alpha \mid acts(A)$ y, además, el *comportamiento* de una ejecución α, $beh(\alpha) = \alpha \mid ext(A)$. De manera análoga se define $scheds(A)$ como el conjunto de planes de ejecuciones de A y $behs(A)$, el conjunto de los comportamientos de las ejecuciones de A. El conjunto de los comportamientos de las ejecuciones equitativas del autómata A, es decir, los comportamientos equitativos se denominan $fairbehs(A)$.

[1] Cada clase de *part(A)* presenta equidad débil en el sistema.

A.1.1. Composición de autómatas

En el modelo existe un operador que permite combinar autómatas y se denomina operador composición. Haciendo uso de este operador se puede analizar la interacción de las diferentes partes de un sistema.

Un conjunto de autómatas $\{A_i |\ i \in I\}$ es compatible si, $\forall\ (i, j) \subseteq I \times I,\ i \neq j$, se cumple que:

- $out(A_i) \cap out(A_j) = \emptyset$,
- $int(A_i) \cap acts(A_j) = \emptyset$,
- no hay ninguna acción compartida por un número infinito de autómatas.

La composición de una colección de autómatas fuertemente compatibles $\{A_i |\ i \in I\}$ se llama A y se define como $A = \prod_{i \in I}\ A_i$. Los componentes de la composición A son:

- $out(A) = \bigcup_{i \in I}\ out(A_i)\ in(A) = \bigcup_{i \in I}\ in(A_i) \setminus out(A)\ int(A) = \bigcup_{i \in I}\ int(A_i)$,
- $states(A) = \prod_{i \in I}\ states(A_i)$,
- $start(A) = \prod_{i \in I}\ start(A_i)$,
- $steps(A)$ es el conjunto de ternas de la forma $(s_{z-1},\ \pi_z,\ s_z)$ tales que $\forall\ i \in I$:
  - si $\pi_z \in acts(A_i)$, entonces $s_{z-1}[\mathrm{A}_i],\ \pi_z,\ \mathrm{s}_z[\mathrm{A}_i] \in steps(A_i)$
  - si $\pi_z \notin acts(A_i)$, entonces $s_{z-1}[A_i] = s_z[A_i]$
- $part(A) = \bigcup_{i \in I}\ part(A_i)$.

Una transición del autómata compuesto a través de una acción π consiste en todos los autómatas que incluyen a π en su signatura ejecutando concurrentemente dicha

acción, mientras que los autómatas componentes que no la tengan no hacen nada. La partición del autómata composición se obtiene de la unión de las particiones de los componentes. De esta forma una ejecución equitativa del autómata compuesto es equitativa con todas las clases de los autómatas componentes.

Los resultados presentados en [114] exponen que la evolución del autómata compuesto no es más que la evolución sincronizada de todos sus componentes. Para enunciar estos resultados se define un nuevo tipo de operador que se denomina operador restricción. Sea $\alpha = s_0\ \pi_1\ s_1 \ldots s_{k-1}\ \pi_k\ s_k \ldots$ una ejecución del autómata composición $A = \prod_{i \in I} A_i$. Se define $\alpha \dagger A_i$ como la secuencia que se obtiene eliminando $\pi_j\ s_j$ de α, si π_j no pertenece a $acts(A_i)$, y sustituyendo en la secuencia resultante s_j por $s_j[i]$.

Lema A.1.1. *Sea* $\{\mathrm{A}_i\}_{i \in I}$ *una colección de autómatas (fuertemente) compatibles y* $A = \prod_{i \in I} A_i$. *Si* α *es una ejecución de* A, *entonces* $\alpha \dagger A_i$ *es una ejecución de* A_i *para todo* $i \in I$. *Este mismo resultado se verifica para ejecuciones equitativas, comportamientos y comportamientos equitativos.*

Lema A.1.2. *Sea* $\{A_i\}_{i \in I}$ *una colección de autómatas compatibles y* $A = \prod_{i \in I} A_i$. *Sea* β *una secuencia de acciones de acts(A). Si* $\beta \mid acts(A_i)$ *es un comportamiento equitativo de* A_i *para todo* $i \in I$, *entonces* β *es un comportamiento equitativo de* A. *Si* $\beta \mid acts(A_i)$ *es un comportamiento de* A_i *que deja* A_i *en el estado* s_i *para todo* $i \in I$, *entonces* β *es un comportamiento de* A *que deja* A *en un estado* s *tal que* $s[i] = s_i$ *para todo* $i \in I$.

A.2. Otros objetos del modelos de autómatas de Entrada/Salida

Además de los autómatas en el modelo de autómatas de Entrada/Salida se definen otro tipo de objetos. A continuación se van a describir los módulos de ejecuciones y los módulos de planes, que son otros objetos del modelo que se han empleado en el diseño del algoritmo de esta tesis.

A.2.1. Módulos de ejecuciones

Un módulo de ejecuciones E está formado por los siguientes elementos:

- la signatura de acciones de E, $sig(E)$,
- el conjunto de estados de E, $states(E)$,
- el conjunto de ejecuciones de E, $execs(E)$.

Cada ejecución de E consiste en una secuencia finita o infinita que, comenzando en un estado, va alternando acciones y estados de E hasta alcanzar, en caso de una ejecución finita, un estado final. Se asume que todas las ejecuciones de E son equitativas, es decir, $fairexecs(E) = execs(E)$. Por otra parte, tanto los conceptos de plan y comportamiento de una ejecución perteneciente a un módulo de ejecuciones como los conceptos de conjunto de planes, $scheds(E)$, y de comportamientos, $fairbehs(E)$ de un módulo de ejecuciones son análogos a los correspondientes conceptos de autómatas.

Se dice que un módulo de ejecuciones E es un módulo de ejecuciones del autómata A si $sig(E) = sig(A)$, $states(E) = states(A)$ y, además, $execs(E) \subseteq execs(A)$. Al módulo de ejecuciones de A que contiene las ejecuciones equitativas de A, $fairexecs(A)$, se le denomina *módulo trivial de ejecuciones* de A y se denota como $Fair(A)$.

Los módulos de ejecuciones se emplean principalmente para restringir el conjunto de ejecuciones de un autómata concreto. A las ejecuciones de un módulo de ejecuciones se les suele exigir que verifiquen un conjunto de propiedades de progreso. Estas propiedades se denotan con predicados de estado del tipo $S \hookrightarrow P$, donde S es un conjunto de estados y P un conjunto de acciones. Si las ejecuciones de un módulo de ejecuciones cumplen $S \hookrightarrow P$, entonces se verifica que a todo estado de S resultante de una ejecución le llega a suceder siempre alguna acción de P.

El módulo trivial de ejecuciones de un autómata A, $Fair(A)$, se puede reescribir usando este tipo de predicados. Para ello basta con definir para cada una de las clases

de equivalencia, C, de $part(A)$ el predicado $S_C \hookrightarrow P_C$, donde S_C es el conjunto que contiene todos los estados en los que hay habilitada alguna acción de C y P_C es la unión de C y el conjunto de todas las acciones que pueden deshabilitar alguna de las acciones de C.

A.2.2. Módulos de planes

Los elementos que constituyen un módulo de planes P son:

- la signatura de acciones de P, $sig(P)$,
- el conjunto de planes de P, $scheds(P)$.

Un plan de P puede ser una secuencia de acciones de P finita o infinita. Tanto el comportamiento de un plan perteneciente a un módulo de planes como el conjunto de comportamientos de un módulo de planes son conceptos análogos a los definidos para autómatas. Por definición se establece que todos los planes de P son equitativos, esto es, $fairbehs(P) = behs(P)$.

Un módulo de planes, P, de un módulo de ejecuciones E, es aquel módulo de planes que verifica que $sig(P) = sig(E)$ y $scheds(P) \subseteq scheds(E)$. Se denomina *módulo trivial de planes* de E, denotado $Scheds(E)$, al módulo de planes de E que cumple la relación $scheds(Scheds(E)) = scheds(E)$.

A.2.3. Especificaciones

Un módulo de planes, P, cuya signatura es una signatura externa de acciones se denomina *especificación*. Si P es una especificación, entonces $int(P) = \emptyset$. Estos módulos de planes especiales reciben el nombre de especificación porque es el objeto del modelo de autómatas de Entrada/Salida que se usa habitualmente para describir o especificar el algoritmo que se pretende desarrollar. Si el módulo de planes P es una especificación, el conjunto $scheds(P)$ contiene las secuencias de acciones de entrada y

salida que permiten definir la relación entre el algoritmo y su entorno sin tener que recurrir a nombrar variables concretas del algoritmo.

En resumen, una especificación es un conjunto de comportamientos permitidos. Así que, un autómata se dice que satisface una especificación cuando cada uno de sus comportamientos está contenido en el conjunto de comportamientos de la especificación.

A.3. Pruebas jerárquicas de corrección

Todos los objetos del modelo de autómatas de Entrada/Salida que se han definido en las secciones previas se caracterizan por tener una gran capacidad descriptiva. En consecuencia, se podrán utilizar objetos tanto para describir la solución deseada a un determinado problema como la solución propuesta. Para la comprobación del objeto de la solución propuesta será preciso disponer de un mecanismo que evalúe si éste verifica las condiciones establecidas en el objeto de la solución deseada. En el modelo de autómatas de Entrada/Salida esta evaluación positiva recibe el nombre de *satisfacción*. Así pues, un objeto A satisface a otro B cuando $in(A) = in(B)$, $out(A) = out(B)$ y $fairbehs(A) \subseteq fairbehs(B)$. En otras palabras, los dos objetos se comunican con el entorno de la misma forma y el conjunto de comportamientos de A es un subconjunto del conjunto de comportamientos de B, por tanto, los comportamientos del objeto A cumplen las mismas propiedades que los comportamientos del objeto B.

Una forma de demostrar que un autómata A verifica las mismas propiedades de seguridad que otro autómata B, es establecer una correspondencia entre los estados de ambos autómatas. Esta correspondencia se denomina *mapa de posibilidades*.

Sean A y B dos autómatas con las mismas acciones externas, $ext(A) = ext(B)$, y sea h una función inyectiva de los estados de A en las partes del conjunto de estados de B (si s es un estado de A, $h(s)$ es un conjunto de estados de B). La función h es un mapa de posibilidades de A en B si cumple los siguientes requisitos:

- Para cada estado s_0 de $start(A)$ existe un estado inicial t_0 de $start(B)$ tal que $t_0 \in h(s_0)$.
- Sea s un estado alcanzable de A, t un estado alcanzable de B con $t \in h(s)$ y (s, π, $s\prime$) un paso de A, entonces se cumple que:
  - Si $\pi \in acts(B)$, entonces (t, π, $t\prime$) $\in steps(B)$ con $t\prime \in h(s\prime)$.
  - Si $\pi \notin acts(B)$, entonces $t \in h(s\prime)$.

La existencia de un mapa de posibilidades de A en B asegura que los comportamientos de A son un subconjunto de los comportamientos de B, es decir, el objeto A *implementa* al objeto B. Por lo tanto, se puede afirmar que el autómata A verifica las mismas propiedades de seguridad que B. Aunque un comportamiento que no pueda darse en B no puede tener lugar en A, no se garantiza que el autómata A verifique las mismas propiedades de viveza que el autómata B.

Una manera de demostrar que un objeto A verifica las mismas propiedades de viveza que otro objeto B es comprobar que el conjunto de comportamientos equitativos de A es un subconjunto de los comportamientos equitativos de B. En el modelo de autómatas de Entrada/Salida esta situación recibe el nombre de *satisfacción*. Un objeto A satisface a otro B si y sólo si $ext(A) = ext(B)$ y $fairbehs(A) \subseteq fairbehs(B)$. Esto implica que los dos objetos se comunican con el entorno de la misma manera y el conjunto de comportamientos equitativos de A es un subconjunto del de B, con lo que los comportamientos equitativos de A cumplen las mismas propiedades que los comportamientos equitativos de B.

Para que un objeto A satisfaga a un objeto B es necesario que se cumpla la relación $fairbehs(A) \subseteq fairbehs(B)$. En [114] se demuestran los teoremas que determinan algunas condiciones en las que algunos objetos satisfacen a otros. El primer teorema hace referencia a la satisfacción de un autómata por otro. Su ámbito de aplicación es reducido ya que precisa que los autómatas sean muy próximos.

Teorema A.3.1. *Sean A y B dos autómatas. Si se cumplen las siguientes condiciones:*

- $part(B) \subseteq part(A)$.
- *Existe un mapa de posibilidades h de A en B.*
- *Para todo estado alcanzable de A, s, y para cada clase $C \in part(A)$ y $D \in part(B)$ tal que $D \subseteq C$ se cumple que si una acción de D está habilitada en un estado alcanzable de $h(s)$, entonces una acción de D está habilitada en el estado s y ninguna acción de C-D está habilitada en ese estado s.*

entonces $fairbehs(A) \subseteq fairbehs(B)$.

El segundo teorema define algunas condiciones en las que un módulo de ejecuciones satisface otro.

Teorema A.3.2. *Sean A y B dos autómatas y h un mapa de posibilidades de A en B. Sea E un módulo de ejecuciones de A cuyas ejecuciones verifican, $\forall\ i \in I$, el conjunto de predicados $S_i \hookrightarrow T_i$. Sea F un módulo de ejecuciones de B cuyas ejecuciones verifican, $\forall\ i \in I$, el conjunto de predicados $U_i \hookrightarrow V_i$. Si $\forall\ i \in I$ se verifica que $h^{-1}(U_i) \subseteq U_i$, entonces E satisface F.*

La utilidad del modelo deriva de las características de la definición de satisfacción. Como consecuencia, los argumentos de las pruebas jerárquicas de corrección (refinamiento por pasos sucesivos) son sustentados por el siguiente hecho: si A satisface B y B satisface C, entonces A satisface C. De este modo, se puede definir una secuencia de objetos O_1,... O_n que define la solución en niveles de abstracción decreciente. Únicamente debe demostrarse que cada modelo O_i satisface O_{i-1}. Este es el concepto de prueba jerárquica de corrección. Obviamente el objeto O_1 será una especificación y el objeto O_n un autómata. De no ser así, el objeto O_n no describiría completamente cómo se resuelve la especificacción.

Apéndice B

Demostraciones de propiedades

Para evitar al lector pérdidas de tiempo, se ha tratado de mostrar tan sólo aquellos aspectos de las demostraciones que puedan ser conflictivos. Es por ello que se han suprimido algunas formalidades. A no ser que se diga lo contrario, las propiedades se demuestran por inducción sobre una ejecución $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots \in execs(G)$. En el paso de inducción se hará referencia sólo a aquellas acciones que modifiquen algunas de las variables de estado a las que se refiere el enunciado de la propiedad. Obviamente, en el resto de casos la hipótesis inductiva concluye.

B.1. Propiedades del medio

En la primera parte de este Apéndice B se presentan algunas propiedades que cumple el autómata del medio G y que son de utilidad para demostrar la corrección del algoritmo.

Las primeras dos propiedades comprueban que la variable *blocker* está acorde con la variable *state* de un nodo. Parece lógico pensar que un nodo está bloqueado si y sólo si tiene *blocker*. Sin embargo, el modo en que se eliminan las esperas tras un aborto no permite asegurar una condición tan fuerte. Sólo es posible asegurar la equivalencia entre la no existencia de blocker y la imposibilidad de que un nodo esté bloqueado.

Propiedad B.1.1. $\forall\ i \in \mathcal{N}$ se verifica que $s.blocker_i = NULL \Leftrightarrow s.state_i \neq blocked$.

Demostración: El estado inicial, s_0, verifica que $\forall\ i \in \mathcal{N}$: $s_0.blocker_i = NULL$ y $s_0.state_i = active$, por lo que se cumple la propiedad. Las únicas acciones que modifican $blocker_i$ o $state_i$ son:

- $\pi_z = Abort_i$. Tras su ejecución, $s_z.blocker_i$ pasa a $NULL$ y $s_z.state_i$ a *aborted*.
- $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$. Tras su ejecución, $s_z.blocker_i = j$ y $s_z.state_i =$ *blocked*.
- $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$. Tras esta acción, $s_z.blocker_i = NULL$ y $s_z.state_i$ $=$ *active*.

■

De igual modo, puede comprobarse que los nodos con *blocker* no pueden ser *active*, aunque no tienen por qué ser *blocked*, también pueden ser *victim*.

Propiedad B.1.2. $\forall\ \{i, j\} \subseteq \mathcal{N}$ se verifica que $s.blocker_i = j \Rightarrow s.state_i \neq active$.

Demostración: El estado inicial, s_0, verifica trivialmente la propiedad pues $\forall\ i \in \mathcal{N}$: $s_0.blocker_i = NULL$. Las acciones que afectan a la propiedad son:

- $\pi_z = Abort_i$ o $\pi_z = EndDelArc_i(j)$. Tras la ejecución de cualquiera de ellas, $s_z.blocker_i = NULL$.
- $\pi_z = StartAddArc_i(j)$. Tras la ejecución de la acción, $s_z.blocker_i = j$ pero $s_z.state_i$ $=$ *blocked*.

■

Las variables *set_waiters* del medio se utilizan para modelar la existencia de mensajes entre los nodos del sistema. La siguiente propiedad indica que cuando un nodo ha sido abortado su *set_waiters* sólo puede contener ternas *sent* en su tercer campo. Es decir, la única posibilidad de que haya algún mensaje dirigido hacia un nodo abortado es que otro nodo pretenda comenzar a esperar por él. Esta situación puede darse si ese otro nodo no sabe aún que el nodo ha abortado.

Propiedad B.1.3. Si $\exists\ \{i, j\} \subseteq \mathcal{N}$, $\exists\ t \in \mathbb{N}$ y $\exists\ b \in \{sent, received, released\}$: $s.state_i$ $=$ *aborted* $\wedge\ (j,\ t,\ b) \in s.set\_waiters_i \Rightarrow b = sent$.

Demostración: Es obvio que el estado inicial, s_0, verifica la propiedad pues $\forall\ i \in \mathcal{N}\ s_0.set\_waiters_i = \emptyset$. Se analizan las acciones que afectan a las variables de la propiedad:

- $\pi_z = Abort_i$. Su ejecución hace que $s_z.set\_waiters_i = \emptyset$.
- $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$. Los efectos de la acción hacen que $s.state_i =$ *blocked*.

- $\pi_z = StartAddArc_j(i)$. La acción sólo incorpora en $s_z.set\_waiters_i$ el elemento $(j,\ s_{z-1}.t\_activ_j,\ sent)$.
- $\pi_z \in \{EndAddArc_i(x,\ t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Como la acción está habilitada, se verifica que $s_{z-1}.state_i \neq aborted$ y su ejecución no modifica $state_i$.
- $\pi_z = StartDelArc_i(j,\ t)$. Esta acción sólo añade elementos a $s_{z-1}.set\_waiters_i$ cuando $s_{z-1}.state_i \neq aborted$. Como su ejecución no modifica la variable $state_i$, la hipótesis inductiva concluye.
- $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$. La acción hace que $s_z.state_i = active$.
- $\pi_z = EndDelArc_j(i)$. La acción nunca añade elementos a $s_{z-1}.set\_waiters_i$.

■

La siguiente propiedad asegura que cuando un nodo, j, tiene información que le permite creer que otro nodo, i, está esperando por él, es porque, bien i está efectivamente esperando por él en ese instante, o bien porque i ha abortado y la información sobre dicho aborto no ha sido aún recibida por j.

Propiedad B.1.4. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$, $\exists\ b \in \{sent,\ received,\ released\} \wedge \exists\ t \in \mathbb{N}$ tales que $(i,\ t,\ b) \in s.set\_waiters_j \Rightarrow (s.state_i = blocked \wedge s.blocker_i = j \wedge s.t\_activ_i = t \wedge \forall\ b\prime \neq b\colon (i,\ t,\ b\prime) \notin s.set\_waiters_j) \vee (s.state_i = aborted \wedge s.t\_activ_i > t)$.

Demostración: El estado inicial, s_0 verifica que $\forall\ i \in \mathcal{N}\colon s_0.set\_waiters_i = \emptyset$, por lo que la propiedad es cierta. Seguidamente se comprueban las acciones que afectan a las variables de la propiedad.

- $\pi_z = Abort_j$. Hace falso el antecedente puesto que deja vacío $s_z.set\_waiters_j$.
- $\pi_z = Abort_i$. Su ejecución no modifica $s_{z-1}.set\_waiters_j$, por tanto $(i,\ t,\ b)$ sólo puede estar en $s_z.set\_waiters_j$ si lo estaba en el estado anterior. En ese caso, la hipótesis inductiva permite asegurar que $s_{z-1}.t\_activ_i \geq t$. Como la acción aumenta el valor de t_activ_i y $s_z.status_i$ pasa a *aborted*, se hace cierto el consecuente.
- $\pi_z = StartAddArc_i(j)$. Si la acción está habilitada se cumple que $s_{z-1}.state_i = active \notin \{blocked,\ aborted\}$, por lo que la hipótesis inductiva asegura que $\nexists\ b,\ t\colon (i,\ t,\ b) \in s_{z-1}.set\_waiters_j$. La acción inserta $(i,\ t\_activ_i,\ sent)$ en $set\_waiters_j$, pero también hace que $s_z.state_i = blocked$ y $s_z.blocker_i = j$. Como no incluye nada más en $set\_waiters_j$, se concluye que la propiedad se cumple en s_z.

- $\pi_z \in \{EndAddArc_j(i,\ t),\ StartDelArc_j(i,\ t)\colon\ t \in \mathbb{N}\}$. Si alguna de estas acciones está habilitada se cumple que $s_{z-1}.set\_waiters_j$ contiene una terna $(i,\ t,\ b)$. La hipótesis inductiva asegura, por tanto, que el consecuente de la propiedad es cierto $(t = s_{z-1}.t\_activ_i)$. Los efectos de ambas acciones se limitan, en todo caso, a cambiar el elemento $(i,\ t,\ b)$ de $s_{z-1}.set\_waiters_j$ por $(i,\ t,\ b\prime \neq b)$ por lo que la propiedad se mantiene.

- $\pi_z = EndDelArc_i(j)$. Si la acción está habilitada se verifica que bien la terna $(i,\ s_{z-1}.t\_activ_i,\ released) \in s_{z-1}.set\_waiters_j$ o bien que $s_{z-1}.state_j = aborted$. En el primer caso, por la hipótesis inductiva, las ternas $(i,\ s_{z-1}.t\_activ_i,\ b)$ con $b \in \{sent,\ received\}$ no pueden estar en $s_{z-1}.set\_waiters_j$ y los efectos de la acción eliminan de dicho conjunto $(i,\ s_{z-1}.t\_activ_i,\ released)$. En el segundo caso, la propiedad B.1.3 indica que sólo la terna $(i,\ s_{z-1}.t\_activ_i,\ sent)$ puede pertenecer a $s_{z-1}.set\_waiters_j$ y la acción la elimina. Por tanto, en s_z no quedan ternas de la forma $(i,\ t,\ b)$ en $s_z.set\_waiters_j$.

■

Esta propiedad recoge un detalle técnico bastante obvio. Cada posible valor del tercer campo de los elementos de *set_waiters* representa una fase de la adición/borrado de una espera. No es posible que una espera se encuentre simultáneamente en más de una fase.

Propiedad B.1.5. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$ verificando que $(i,\ t,\ b) \in s.set\_waiters_j \Rightarrow \forall\ b\prime \neq b\colon (i,\ t,\ b\prime) \notin s.set\_waiters_j$.

Demostración: El estado inicial s_0 verifica que, $\forall\ j \in \mathcal{N}$, $s_0.set\_waiters_j = \emptyset$, por lo que la propiedad es cierta.

- $\pi_z = Abort_j$. Los efectos de la acción hacen el antecedente falso, $s_z.set\_waiters_j = \emptyset$.

- $\pi_z = StartAddArc_i(j)$. Los efectos de la acción hacen el antecedente de la propiedad cierto porque $(i,\ t\_activ_i,\ sent) \in s_z.set\_waiters_j$. Sabiendo que otros efectos de la acción son: $s_z.state_i = blocked$ y $s_z.blocker_i = j$, se aplica la propiedad B.1.4 que confirma que, en la situación analizada, ni $(i,\ t\_activ_i,\ received)$ ni $(i,\ t\_activ_i,\ released)$ pertenecen a $s_z.set\_waiters_j$. Por consiguiente, la propiedad se verifica.

- $\pi_z \in \{EndAddArc_j(i,\ t)\colon\ t \in \mathbb{N}\}$. La acción está habilitada cuando $(i,\ t,\ sent) \in s_{z-1}.set\_waiters_j$. La hipótesis inductiva establece que en s_{z-1} las tuplas $(i,\ t,\ received)$ y $(i,\ t,\ released)$ no forman parte de $set\_waiters_j$. Al ejecutarse la

acción, se sustituye la tupla de valor *sent* por una nueva tupla con el valor *received*. Así que, tras producirse este efecto, la propiedad se cumple.

- $\pi_z \in \{StartDelArc_j(i,\ t)\colon\ t \in \mathbb{N}\}$. Para que la acción esté habilitada se requiere que $(i,\ t,\ received) \in s_{z-1}.set\_waiters_j$. Según la hipótesis inductiva, esto implica que ni $(i,\ t,\ sent)$ ni $(i,\ t,\ released)$ están contenidos en $s_{z-1}.set\_waiters_j$. En el caso de que $s_{z-1}.state_j$ sea *active*, al ejecutarse la acción se sustituye la tupla existente por $(i,\ t,\ released)$. Este efecto hace que la propiedad se cumpla con antecedente y consecuente ciertos. En en el otro caso posible de ejecución, $s_{z-1}.state_i$ es *aborted* y los efectos de la acción tan sólo eliminan la tupla $(i,\ t,$ *received*$)$ del conjunto $set\_waiters_j$. En esta situación, la propiedad se cumple en s_z con antecedente falso.

- $\pi_z = EndDelArc_i(j)$. Si la acción se ejecuta con $(i,\ t,\ released) \in s_{z-1}.set\_waiters_j$, la hipótesis inductiva establece que ni la tupla $(i,\ t,\ sent)$ ni la tupla $(i,\ t,$ *received*$)$ están incluidas en $s_{z-1}.set\_waiters_j$. Los efectos de la acción eliminan el elemento $(i,\ t,\ released)$ de $set\_waiters_j$, pero no incorporan ninguno en este caso. Por tanto, la propiedad se cumple ya que el antecedente se convierte en falso. Cuando la acción está habilitada debido a que $s_{z-1}.state_j = aborted$, la propiedad B.1.3 indica que $(i,\ t,\ sent) \in s_{z-1}.set\_waiters_j$. De acuerdo con la hipótesis inductiva, las tuplas $(i,\ t,$ *received*$)$ y $(i,\ t,\ released)$ no están incluidas en $s_{z-1}.set\_waiters_j$. En este caso, al ejecutarse la acción se elimina $(i,\ t,\ sent)$ del conjunto $set\_waiters_j$ sin incorporar ningún nuevo elemento. En consecuencia, la propiedad se sigue cumpliendo pero con antecedente falso. Por último, cabe mencionar la posibilidad de que el antecedente de la propiedad ya fuera falso en s_{z-1}. Como la acción no modifica la variable $set\_waiters_j$ en esa situación, se llega a la conclusión de que la propiedad se cumple por hipótesis inductiva

■

La siguiente propiedad indica que el autómata G verifica una de las características del modelo que se aborda en esta tesis. En esencia afirma que un nodo que aborta no puede volver a participar de las evoluciones del sistema.

Propiedad B.1.6. Sea $\alpha = s_0.\pi_1.s_1 \ldots \pi_z.s_z \ldots$ una ejecución de G, $\alpha \in execs(G)$. $\forall\ i \in \mathcal{N}$ se verifica que $s_z.state_i = aborted \Rightarrow \forall\ k\colon\ k > z,\ s_k.state_i = aborted$.

Demostración: Por reducción al absurdo. Las únicas acciones que pueden cambiar la variable $state_i$ son $StartAddArc_i(j)$ y $EndDelArc_i(j)$. Para que $StartAddArc_i(j)$ esté habilitada, $state_i$ debe ser *active*. En el caso de la acción $EndDelArc_i(j)$, su precondición obliga a que $blocker_i$ sea j, pero la propiedad B.1.1 indica que entonces $state_i$ debe ser *blocked*. Así que, si $s_z.state_i = aborted$, ninguna de las acciones citadas podrá estar habilitada. ■

Esta última propiedad del medio corrobora que el autómata G representa correctamente la características básica del modelo *Single Request* impuesto, esto es, un nodo no puede esperar simultáneamente por más de un nodo.

Propiedad B.1.7. Sea $\alpha = s_0.\pi_1.s_1 \ldots \pi_z.s_z \ldots$ una ejecución de G, $\alpha \in execs(G)$. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$, $\exists\ b \in \{sent,\ received,\ released\}$ y $\exists\ t \in \mathbb{N}$ tales que $(i,\ t,\ b) \in s_z.set\_waiters_j$ entonces se verifica que $\forall\ z\prime \in \mathbb{N}\ (z\prime \neq z)$ y $\forall\ k \neq j$: $(i,\ t,\ b\prime) \notin s_{z\prime}.set\_waiters_k$.

Demostración: Por reducción al absurdo. Supóngase que $(i,\ t,\ b) \in s_z.set\_waiters_j$ y que $\exists\ z\prime$ tal que $(i,\ t,\ b) \in s_{z\prime}.set\_waiters_k$ siendo $k \neq j$ y $z\prime > z$ (el caso contrario, $z\prime < z$, se trataría de forma simétrica). La acción que hace aparecer inicialmente una terna $(i,\ t,\ b\prime)$ en $set\_waiters_k$ es $StartAddArc_i(k)$ que tiene como precondición $state_i = active$. Pero por la propiedad B.1.4, como $(i,\ t,\ b) \in s_z.set\_waiters_j$ entonces $(t = s_z.t\_activ_i \wedge s_z.state_i = blocked) \vee s_z.state_i = aborted$. En el primer caso, $state_i$ debe pasar a ser *active*. La acción que hace posible este cambio es $EndDelArc_i(j)$. Los efectos de esta acción incluyen un incremento del tiempo de activación del nodo i. Esto implica que, si el nodo i pasa a pertenecer a $set\_waiters_k$, obligatoriamente lo hará con un tiempo distinto al que tenía cuando pertenecía a $set\_waiters_j$. En el segundo caso, la propiedad B.1.6 asegura que jamás se podrá ejecutar $\pi_{z\prime} = StartAddArc_i(k)$. ■

B.2. Propiedades del sistema *S*

En esta segunda parte del apéndice B se proporcionan las propiedades que verifica el autómata S, resultado de la composición del autómata del medio, G, con el autómata del algoritmo, A.

En el capítulo 3 se definió el conjunto P como el conjunto que contiene todos los pares (*nodo*, *tiempo*) que pueden aparecer durante la ejecución del algoritmo. Los elementos de P van a jugar un papel capital en la demostración del algoritmo puesto que son los elementos constitutivos de las rutas.

Los elementos de P se almacenan en diversas variables durante la ejecución del algoritmo. Cabe recordar que cada par (*nodo*, *tiempo*) tiene que ver con la creación de una nueva espera por parte del *nodo*. Teniendo en cuenta esto, parece obvio que, en un determinado momento de la ejecución, los elementos de P almacenados en variables del algoritmo deben cumplir la condición de que su componente temporal sea menor o igual al tiempo local del *nodo*. Para poder expresar esta propiedad se

van a definir previamente algunos conceptos que permitirán aligerar la notación del resto del trabajo.

En primer lugar se define una aserción que va a indicar que una ruta está recogida en alguna de las variables del algoritmo.

Definición B.2.1. Dada una ruta $p \in P^+$ se dice que está almacenada en un estado $s \in states(S)$ y se denota $recordedPath(p, s)$ si y sólo si

- $\exists\ i \in \mathcal{N}$, $sid \in T$: $(sid, p) \in s.st\_alg_i \vee$
- $\exists\ i \in \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$: $(t, sid, p) \in s.st\_avsrsp_i \vee$
- $\exists\ i \in \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $b \in \{true, false\}$: $(sid, t, p, b) \in s.set\_st\_avs_i \vee$
- $\exists\ \{i, j\} \subseteq \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$: $(ALG, t, sid, p) \in s.channel(i, j) \vee$
- $\exists\ \{i, j\} \subseteq \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$: $(AVSRSP, t, sid, p) \in s.channel(i, j) \vee$
- $\exists\ \{i, j\} \subseteq \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $b \in \{true, false\}$: $(AVS, sid, t, p, b) \in s.channel(i, j)$

A continuación se define el conjunto de elementos de P que está almacenado en alguna variable en un momento dado de la ejecución del algoritmo. Cada uno de esos pares representa una espera que el algoritmo considera que puede existir.

Definición B.2.2. Se denota como $recordedWaits(s)$, siendo s un estado alcanzable del sistema, al subconjunto de elementos de P almacenados en alguna variable en el estado s. Es decir, $recordedWaits(s) = \{(i, t) \in P$ tales que:

- $\exists\ j \in \mathcal{N}$: $(i, t) \in s.setPred_j$
- $\exists\ j \in \mathcal{N}$: $(i, t) \in s.setPredToInf_j$
- $\exists\ p \in P^+$ tal que $recordedPath(p, s)$ e $(i, t) \in visited\_nodes(p)$ $\}$

Utilizando las definiciones anteriores, la formulación de las siguientes propiedades relacionadas con el tiempo queda simplificada. Toda espera de la que el algoritmo ha tomado nota en alguna de sus variables comenzó en un momento anterior. Por consiguiente, el temporizador local del nodo al que hace referencia puede haber aumentado.

Propiedad B.2.1. Se verifica que $\forall\ (i,\ t) \in P$ tal que $(i,\ t) \in recordedWaits(s) \Rightarrow 1 \leq t \leq s.ta_i$.

Demostración: En el estado inicial, s_0, se verifica que $\forall\ \{i,\ j\} \subseteq \mathcal{N}$: $s_0.setPred_j = s_0.setPredToInf_j = s_0.set\_st\_avs_j = \emptyset$. Además, $s_0.st\_alg_j = s_0.st\_avsrsp_j = NULL$, $s_0.channel(i,\ j) = \epsilon$ y $s_0.ta_i = 1$. Los valores iniciales de todas estas variables hacen que la implicación sea cierta.

Para demostrar la propiedad, se va a analizar en primer lugar cómo influyen los cambios en el tiempo de activación de los nodos y después se verificará la condición temporal para todos los pares (*nodo*, *tiempo*) que están presentes en cualquiera de las variables que constituyen la definición de $recordedWaits(s_z)$.

Los efectos de las acciones $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ y $\pi_z = Abort_i$ modifican el tiempo de activación del nodo i, de manera que $s_{z-1}.ta_i < s_z.ta_i$. Así que, para cualquier par (i,t) que pudiera estar registrado en $recordedWaits(s_{z-1})$, la hipótesis inductiva indica que $1 \leq t \leq s_{z-1}.ta_i$. Tras ejecutarse cualquiera de estas dos acciones, la propiedad se cumple porque $1 \leq t < s_z.ta_i$.

Por otra parte, la ejecución de la mayor parte de las acciones mantiene sin cambios los pares (*nodo*, *tiempo*) contenidos en cualquiera de las variables que componen *recordedWaits*, esto es, si $(i,\ t)$ pertenece a $recordedWaits(s_z)$ es porque, en el estado previo, $(i,\ t)$ ya constaba en $recordedWaits(s_{z-1})$. Por tanto, basta con comprobar que las acciones que incluyen nuevos pares $(i,\ t)$ en $recordedWaits(s_z)$, sin que $(i,\ t)$ estuviera registrado en $recordedWaits(s_{z-1})$, lo hacen verificando que $1 \leq t \leq s_z.ta_i$.

En lo que respecta a las rutas de nuevos mensajes almacenados o en el canal de comunicaciones, las acciones $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = initiate_i$, $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z = \{firstAVS_i\}$ incorporan el par $(i,s_z.ta_i)$ en la ruta de los mensajes que pueden formarse por sus efectos. Dado que los efectos de estas acciones no modifican la variable ta_i, se cumple, aplicando la hipótesis inductiva, que $s_{z-1}.ta_i = s_z.ta_i$. Por tanto, la propiedad se hace evidente porque inicialmente el tiempo de activación de cualquier nodo del sistema vale la unidad, $1 \leq s_z.ta_i$.

Considerando la variable *setPred*, tan sólo la acción $\pi_z \in \{EndAddArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ añade elementos al conjunto $setPred_i$. Al ejecutarse esta acción, $(j,\ t) \in s_z.setPred_i$. Aplicando la propiedad del B.3.2 se tiene que, equivalentemente, $(j,\ t,\ received) \in s_z.set\_waiters_i$. De acuerdo con la propiedad del medio B.1.4, se sabe que $s_z.t\_activ_j \geq t$. En ambos casos la propiedad B.3.1, establece la equivalencia $s_z.t\_activ_j = s_z.ta_j$. Esto implica que el par que se incluye en el conjunto es $(j,s_z.ta_j)$. Reescribiendo las relaciones, resulta $t \leq s_z.ta_j$. Además, según la definición de la acción t toma sus valores del conjunto $\mathbb{N}$ y el tiempo de activación inicial del nodo j es la unidad, lo que implica que $1 \leq t$. Por lo tanto, la propiedad es cierta, $1 \leq t \leq s_z.ta_j$.

Finalmente, en el conjunto $setPredToInf_i$ puede incorporarse un nuevo par (*nodo*,*tiempo*) por efecto de la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. Como el elemento se añade sólo si pertenecía a $setPred_i$ previamente y la variable $setPred_i$ no se ve modificada en esas acciones, la hipótesis inductiva confirma que la propiedad se cumple en estas situaciones. ■

El tiempo del mensaje o del nodo al que va destinado cualquiera de los mensajes presentes en los canales de comunicación está acotado. En la siguiente propiedad se va a demostrar que el tiempo del mensaje puede tomar valores naturales que van desde la unidad hasta el tiempo de activación vigente del nodo destino.

Propiedad B.2.2. $\forall \{i, j\} \subseteq \mathcal{N}$ se verifica que $\forall\ m \in s.channel(j, i)\colon m.type \in \{ALG, AVS, AVSRSP, INF\} \Rightarrow 1 \leq m.ta \leq s.ta_i$.

Demostración: En el estado inicial, s_0, se verifica que $\forall \{i, j\} \subseteq \mathcal{N}\colon s_0.channel(j, i) = \epsilon$. Por tanto, la implicación se cumple. Seguidamente se analizan los efectos de las acciones que afectan a la demostración de la propiedad porque generan mensajes y/o modifican la variable *ta*.

En la generación de cualquier tipo de mensaje se selecciona como tiempo asociado al mensaje el correspondiente a un par (id, t) que forma parte de $recordedWaits(s_{z-1})$. Como los pares incluidos en cualquiera de las variables que constituyen *recordedWaits* cumplen la propiedad B.2.1 y los efectos de las distintas acciones no modifican esos pares, la hipótesis inductiva permite demostrar que el par que se elige como destino de un mensaje verifica $1 \leq t \leq s_z.ta_{id}$ y, por tanto, $1 \leq m.ta \leq s_z.ta_{id}$.

Los mensajes *ALG* que se crean por efecto de las acciones $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z = initiate_i$, $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ fijan su destino a partir de un par almacenado en $s_{z-1}.setPred_i$. La acción $\pi_z \in \{EndAddArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ que también puede formar mensajes *ALG* se sirve del tiempo de un par que incorpora previamente la misma acción en $s_z.setPred_i$. En lo que respecta a la formación de mensajes *AVS*, el tiempo asociado al mensaje se extrae del del penúltimo elemento de una ruta almacenada en $s_{z-1}.set\_st\_avs_i$ o incluida en un mensaje *AVS* del canal en s_{z-1} ($\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$). Otras acciones que en las que se crean mensajes *AVS* son: $\pi_z = firstAVS_i$ y $\pi_z = sndAVS_i$. El tiempo de los mensajes en estos casos queda determinado por el último elemento de la ruta almacenada en st_alg_i (acción $firstAVS_i$) y en st_avsrsp_i (acción $sndAVS_i$). Del mismo modo, los mensajes *AVSRSP* originados tras la ejecución de las acciones: $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$, $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z = sndAVSRSP_i$ se construyen a partir de mensajes almacenados en $s_{z-1}.set\_st\_avs_i$ o procedentes del canal de comunicaciones. El tiempo del nodo

destino que se incluye como parte de estos mensajes *AVSRSP* proviene del primer par registrado en las rutas correspondientes. Por último, al ejecutarse las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, se pueden formar mensajes *INF*. Este tipo de mensajes se dirigirán a nodos que ya estén incluidos en $s_{z-1}.setPredToInf_i$.

Si se consideran los cambios que experimenta la variable *ta*, las únicas acciones que la modifican son $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ y $\pi_z = Abort_i$. Los efectos de ambas incrementan su valor en una unidad, de manera que $s_{z-1}.ta_i < s_z.ta_i$. En s_{z-1}, la hipótesis inductiva establece que, para cualquier tipo de mensaje dirigido al nodo i, $1 \leq m.ta \leq s_{z-1}.ta_i$. Al ejecutarse cualquiera de estas acciones, la propiedad sigue cumpliéndose ya que $1 \leq m.ta < s_z.ta_i$. ■

Tal y como indica el modelo de petición de único recurso, un nodo sólo puede estar bloqueado por otro nodo. Esto implica que un nodo sólo puede estar contenido en el conjunto de predecesores de un único nodo. La propiedad además indica que, a partir de un estado alacanzabe de S, un par concreto (nodo, tiempo) no puede ser más que elemento del conjunto *setPred* de un nodo.

Propiedad B.2.3. Sea $\{i,\ j,\ k\} \subseteq \mathcal{N}$ se verifica que $(i,\ t) \in s.setPred_j \Rightarrow \forall\ z\prime \wedge \forall\ k \neq j\colon (i,\ t) \notin s_{z\prime}.setPred_k$.

Demostración: Por reducción al absurdo. Supóngase que el par $(i,\ t)$ está incluido tanto en el conjunto $s_z.setPred_j$ como en el conjunto $s_z.setPred_k$, siendo $k \neq j$. Aplicando la propiedad B.3.2, se obtiene que la tupla $(i,\ t,\ received)$ pertenece a $s_z.set\_waiters_j$ y a $s_z.set\_waiters_k$ a la vez. Finalmente, considerando la propiedad B.1.7 se llega a la conclusión de que si $(i,\ t,\ received) \in s_z.set\_waiters_j$ entonces, $\forall\ b\prime$ se cumple que $(i,\ t,\ b\prime)$ ya no puede pertenecer al conjunto *set_waiters* en ningún estado, incluido s_z, de otro nodo distinto al nodo j. Es evidente que esto contradice al supuesto de partida por el que $(i,\ t,\ received)$ pertenece a $s_{z\prime}.set\_waiters_k$. ■

De acuerdo con la siguiente propiedad, cuando un nodo conoce su identidad simulada, los elementos de su conjunto de predecesores también deben ser elementos del conjunto de predecesores a los que informar en caso del borrado de la relación de espera.

Propiedad B.2.4. Sea $\{i,\ j\} \subseteq \mathcal{N}$ se verifica que $(i,\ t) \notin s.setPred_j \wedge s.status.id_j = known \Rightarrow (i,\ t) \notin s.setPredToInf_j$.

Demostración: En el estado inicial, s_0, $\forall\ i\colon s_0.setPredToInf_i = \emptyset$, por lo que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$ o $\pi_z = initiate_j$. La ejecución de cualquiera de estas acciones no modifica las variables $setPred_j$ y $status.id_j$. Si $(i,\ t)$ se incorpora a $s_z.setPredToInf_j$ es preciso que $(i,\ t) \in s_{z-1}.setPredToInf_j$. Según esto la propiedad se cumple en s_z.

- $\pi_z \in \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$. Uno de los posibles efectos de la acción consiste en incluir $(i,\ t)$ tanto en $s_z.setPred_j$ como en $s_z.setPredToInf_j$. Es evidente que en esta situación la propiedad se cumple. Si la ejecución de la acción sólo provoca que $(i,\ t) \in s_z.setPred_j$, el antecedente se convierte en falso en s_z.

- $\pi_z \in \{StartDelArc_j(i,\ t)\colon t \in \mathbb{N}\}$. Según las condiciones en las que se ejecute la acción, los efectos pueden retirar el elemento $(i,\ t)$ tanto de $setPred_j$ como de $setPredToInf_j$ o únicamente de $setPred_j$. En el primer caso es obvio la propiedad se cumple en s_z. Cuando se produce el efecto del segundo caso, $s_{z-1}.status.alg_j = active$ y $s_{z-1}.status.id_j = unknown$, resulta que $(y,\ t) \in s_z.setPredToInf_j$ y $status.id_j$ no cambia de valor. En esta situación la propiedad también se verifica en s_z.

- $\pi_z \in \{EndDelArc_j(x)\colon x \in \mathcal{N}\}$. Los efectos de la acción pueden convertir $s_z.status.id_j$ en $unknown$ (antecedente falso).

- $\pi_z = Abort_j$. Tras la ejecución de la acción, $s_z.setPred_j = s_z.setPredToInf_j = \emptyset$ y $s_z.status.id_j = known$. Esto implica que la propiedad se cumple en s_z.

- $\pi_z \in \{rcvALG_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$. Considerando el caso en el que $s_{z-1}.status.alg_j = blocked$, es posible añadir un elemento a $setPredToInf_j$, haciendo el consecuente falso. Para que tenga lugar este efecto es preciso que el elemento esté incluido en $s_{z-1}.setPred_j$ (antecedente falso en s_{z-1}). Los efectos no introducen cambios en las variables, por lo que la propiedad seguirá verificándose en s_z.

- $\pi_z \in \{rcvINF_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. La condición de habilitación de la acción indica que $s_{z-1}.status.id_j = unknown$. Como los efectos de la acción no añaden elementos a $setPred_j$ y por hipótesis inductiva se sabe que $(i,\ t) \notin s_{z-1}.setPred_j$, se concluye que la propiedad es cierta. Sin embargo, cuando $(i,\ t) \notin s_{z-1}.setPred_j$, hay que analizar la variable $setPredToInf_j$, ya que la ejecución de la acción hace que $s_z.status.id_j = known$. Si $(i,\ t) \in s_{z-1}.setPredToInf_j$, el par se elimina del conjunto y el consecuente se convierte en cierto. Por el contrario, si $(i,\ t) \notin s_{z-1}.setPredToInf_j$, el consecuente sigue siendo cierto en s_z. Aunque existe un efecto que incorpora elementos a $setPredToInf_j$, es necesario que éstos estén incluidos en $s_{z-1}.setPred_j$ y este supuesto es imposible.

■

En esta propiedad se afirma que todo par almacenado en el conjunto *setPredToInf* de un nodo está también en el conjunto *setPred* de ese mismo nodo o lo ha estado previamente.

Propiedad B.2.5. Sea $\{i, j\} \subseteq \mathcal{N}$ se verifica que $(i, t) \in s_z.setPredToInf_j \Rightarrow \exists\ z\prime \leq z$: $(i, t) \in s_{z\prime}.setPred_j$.

Demostración: Por reducción al absurdo. Supóngase que el par (i, t) no pertenecía a $setPred_j$ antes de incorporarse a $s_z.setPredToInf_j$, es decir, $\forall\ z\prime \leq z$: $(i, t) \notin s_{z\prime}.setPred_j$. Las acciones que añaden elementos a *setPredToInf* son $StartAdd\ Arc_j(x, t)$, $EndAddArc_j(x, t)$, $initiate_j$, $rcvALG_j(x, m)$ y $rcvINF_j(x, m)$. En el caso de $StartAddArc_j(x, t)$ y $rcvINF_j(x, m)$, la condición para incluir a (i, t) en $s_z.setPredToInf_j$ es que $(i, t) \in s_{z-1}.setPred_j$. Esto entra en contradicción con el supuesto. Los efectos de la acción $EndAddArc_j(x, t)$ añaden al mismo tiempo el elemento (i, t) a los dos conjuntos.Por ese motivo, la propiedad se verifica. Sin embargo, las acciones $initiate_j$ y $rcvALG_j(x, m)$ consiguen que los conjuntos se igualen. Es evidente que en esta situación todos los elementos existentes en $s_{z-1}.setPred_j$ pasan a formar parte de $s_z.setPredToInf_j$. Así que, la suposición de partida es falsa y la propiedad cierta. ■

Cuando un nodo envía un mensaje *INF* a otro para transmitirle información de la identidad simulada, el nodo destino debe estar incluido en el conjunto de *setPredToInf* del nodo emisor y se elimina del conjunto al generarse el mensaje. La propiedad señala que si *setPredToInf* contiene un par (nodo, tiempo) es imposible que haya un mensaje *INF* dirigido a ese nodo.

Propiedad B.2.6. Sea $\{i, j\} \subseteq \mathcal{N}$ se verifica que $(i, t) \in s.setPredToInf_j \Rightarrow \nexists\ m \in M_{INF}$: $m \in s.channel(j, i) \wedge m.ta = t$.

Demostración: En todos los casos en los que se genera un mensaje *INF*, m, se cumple que $(id, t) \in s_{z-1}.setPredToInf_x$. Una vez que se forma este mensaje, se cumple que: $m \in s_z.channel(x, id)$, $m.ta = t$ y, además, $(id, t) \notin s_z.setPredToInf_x$.

Si se supone que $(i, t) \in s_z.setPredToInf_j$ y, a la vez, existe un mensaje *INF* procedente del nodo j al nodo i en el que $m.ta = t$, resulta imposible que tanto el mensaje *INF* como el nodo i incluido en $setPredToInf_j$ tengan asociado el mismo tiempo. Cuando se forma el mensaje, se elimina el elemento (i, t) de *setPredToInf*. Si el nodo i volviera a pertenecer a ese conjunto lo tendría que hacer con otro tiempo tal que $t\prime \neq t$. ■

Esta propiedad completa a la anterior porque asegura que si un par (nodo, tiempo) pertenece al conjunto *setPredToInf* o pertenece al conjunto *setPred* de otro nodo o ese nodo es el destino de un mensaje *INF* entonces ese mismo par no podrá estar en

setPredToInf o en *setPred* de otro nodo distinto o ser el destino de un mensaje *INF* originado por un nodo diferente.

Propiedad B.2.7. Sea $\{i, j\} \subseteq \mathcal{N}$ se verifica que $(i,t) \in s.setPredToInf_j \vee (i,t) \in s.setPred_j \vee \exists\ m \in M_{INF}$: $m \in s.channel(j, i) \wedge m.ta = t \Rightarrow \forall\ k \neq j$, $(i, t) \notin s.setPred_k \wedge (i, t) \notin s.setPredToInf_k \wedge \nexists\ m \in M_{INF}$: $m \in s.channel(k, i) \wedge m.ta = t$.

Demostración: En el estado inicial, s_0, se verifica que $\forall\ \{i, j\} \subseteq \mathcal{N}$: $s_0.setPred_i = s_0.setPredToInf_i = \emptyset$ y $s_0.channel(j, i) = \epsilon$. Por tanto, la propiedad es cierta. A continuación, se analizan los efectos de las acciones que generan mensajes *INF* y/o modifican las variables que aparecen en la propiedad.

- $\pi_z \in \{StartAddArc_j(x)$: $x \in \mathcal{N}\}$. Si $id = i$ y el par $(i, t) \in s_{z-1}.setPred_j \setminus s_{z-1}.setPredToInf_j$, la hipótesis inductiva confirma que tanto el antecedente como el consecuente son ciertos en s_{z-1}. Tras la ejecución de la acción en este caso, $(i, t) \in s_z.setPredToInf_j$, lo que hace que el antecedente siga siendo cierto en s_z. El consecuente también será cierto en s_z porque las variables asociadas al nodo k no cambian y tampoco se forma ningún mensaje *INF* con origen el nodo k.

- $\pi_z \in \{EndAddArc_x(y, t)$: $\{x, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. La precondición de esta acción indica que $(y, t, sent) \in s_{z-1}.set\_waiters_x$. Si $t = s_{z-1}.t\_activ_y$, la propiedad B.1.4 señala que $s_{z-1}.blocker_y = x$. Eso significa que $\forall\ k \neq x$ se tiene que $s_{z-1}.blocker_y \neq k$ y, por consiguiente, $(y, t) \notin s_{z-1}.setPred_k$.

  Por otra parte, se debe demostrar que, si existe un mensaje *INF* en s_{z-1} procedente del nodo $k \neq x$ y dirigido al nodo y, su tiempo será inferior al que posee el nodo y en s_{z-1}. En caso de que existiera un mensaje *INF*, m, tal que $m \in s_{z-1}.channel(k, y)$ y $m.ta = t = s_{z-1}.t\_activ_y$, se deduce que, cuando se formó el mensaje, $(y, t) \in s_{z_1}.setPredToInf_k$, siendo $z_1 < z{-}1$. Considerando la propiedad B.2.5, entonces $\exists\ z_2 \leq z_1$ tal que $(y, t) \in s_{z_2}.setPred_k$ o, lo que es lo mismo, por la propiedad B.3.2, $(y, t, received) \in s_{z_2}.setPred_k$. Según la propiedad B.1.4 se cumple que $s_{z_2}.blocker_y = k$ y $t = s_{z_2}.t\_activ_y$.

  Para tener habilitada la acción es necesario que se ejecute una serie de acciones antes de s_{z-1}. La acción $StartAddArc_x(y)$ en s_{z_5}, siendo $z_5 \leq z{-}1$, permite que el nodo y se bloquee por el nodo x. La condición de habilitación de $StartAddArc_x(y)$ precisa que $s_{z_5-1}.state_y = active$ por eso es necesario que previamente se ejecuten las acciones $\pi_{z_4} = StartDelArc_k(y, t)$ y $\pi_{z_3} = EndDelArc_y(k)$, siendo $z_1 \geq z_4 > z_2$ y $z{-}1 > z_3 > z_4$. Los efectos de la acción $EndDelArc_y(k)$ incrementan el tiempo de activación del nodo y de manera que $t\prime = s_{z_3}.t\_activ_y > s_{z_2}.t\_activ_y = t$. Por tanto, cuando se ejecuta la acción $StartAddArc_x(y)$, el elemento que se incorpora a $set\_waiters_x$ no es $(y, t, sent)$ sino $(y, t\prime, sent)$. En

resumen, no es posible que exista un mensaje *INF*, siendo $k \neq x$, tal que $m \in s_{z-1}.channel(k, y)$ y $m.ta = t = s_{z-1}.t\_activ_y$.

Así mismo, hay que comprobar que $(y, t) \notin s_{z-1}.setPredToInf_k$, siendo $k \neq x$. Si se supone que $(y, t) \in s_{z-1}.setPredToInf_k$, la propiedad B.2.5 establece que $\exists\ z_1 \leq z-1$: $(y, t) \in s_{z_1}.setPred_k$. De acuerdo con la propiedad B.3.2, esto equivale a decir que $(y, t, received) \in s_{z_1}.set\_waiters_k$. La condición de habilitación de la acción indica que $(y, t, sent) \in s_{z-1}.set\_waiters_x$. Así que, para poder ser ejecutada, se han debido suceder las siguientes acciones: $StartAddArc_k(y, t)$ en s_{z_2} con $z_1 < z_2$, $EndDelArc_y(k)$ en s_{z_3}, siendo $z_2 < z_3 < z-1$ y $StartAddArc_x(y)$ en s_{z_4} con $z_3 < z_4 < z-1$. Entre los efectos de la acción $EndDelArc_y(k)$ se incluye el incremento del tiempo de activación del nodo y tal que $t\prime = s_{z_3}.t\_activ_y > s_{z_1}.t\_activ_y = t$. En consecuencia, el elemento que puede incorporarse a $set\_waiters_x$ sólo puede ser $(y, t\prime, sent)$ con lo que la acción no está habilitada. Es evidente, por tanto, que si se ejecuta la acción, $(y, t) \notin s_{z-1}.setPredToInf_k$. De todo ello, se desprende que el consecuente de la propiedad es cierto en s_{z-1}. Tras la ejecución de la acción, el consecuente sigue siendo cierto porque no se generan mensajes *INF* y las variables relacionadas con el nodo $k \neq x$ no sufren cambios.

Según la propiedad B.1.4, también es posible que $(y, t, sent) \in s_{z-1}.set\_waiters_x$ porque $t < s_{z-1}.t\_activ_y$ y $s_{z-1}.state_y = aborted$. Esta situación es imposible que se produzca ya que la única acción que incluye $(y, t, sent)$ en $set\_waiters_x$ es $StartAddArc_y(x)$ y para ello el nodo y debe ser activo. De acuerdo con la propiedad B.1.6, un nodo en estado *aborted* no puede modificar su valor.

- $\pi_z \in \{StartDelArc_x(y, t)\colon \{x, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. La propiedad se cumple en s_{z-1} con antecedente y consecuente ciertos porque $(y, t) \in s_{z-1}.setPred_x$ y en algún caso también $(y, t) \in s_{z-1}.setPredToInf_x$. Al ejecutarse la acción el antecedente podría llegar a ser falso, pero las variables que se incluyen en el consecuente no cambian y no se crea ningún mensaje con origen el nodo k, siendo $k \neq x$.

  Un posible efecto de esta acción es la formación de un mensaje *INF*. Si $x = j$ y $y = i$, se tiene en ese caso que $\exists\ m \in M_{INF}$ tal que $m \in s_z.channel(x, y)$ y $m.ta = t$. La condición para que surja este mensaje es que $(y, t) \in s_{z-1}.setPredToInf_x$ (antecedente y consecuente ciertos en s_{z-1}). Al generarse el mensaje *INF*, el antecedente y el consecuente siguen siendo ciertos.

- $\pi_z = Abort_j$. Al ejecutarse la acción, $s_z.setPred_j = s_z.PredToInf_j = \emptyset$. Otro efecto posible es la generación de mensajes *INF*. Suponiendo que $id = i$, si $(i, t) \in s_{z-1}.setPredToInf_j$ se genera un mensaje *INF*, m, dirigido al nodo i con $m.ta = t$. Como el antecedente es cierto en s_{z-1} porque $(i, t) \in s_{z-1}.setPredToInf_j$, la hipótesis inductiva permite afirmar que el consecuente también es cierto en s_z. Como $k \neq x$, los efectos de la acción no modifican los conjuntos $setPred_k$,

$PredToInf_k$ y no generan ningún mensaje *INF* con origen el nodo k. Esto implica que el consecuente sigue siendo cierto en s_z.

- $\pi_z = initiate_j$ o $\pi_z \in \{rcvALG_j(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{ALG}\}$. Uno de los posibles efectos de estas acciones consiste en igualar el conjunto $setPredToInf_j$ al conjunto $setPred_j$. Si $(id,\ t) \in s_{z-1}.setPred_j$ (antecedente cierto), después de ejecutar la acción $(id,\ t)$ pasa también a formar parte de $s_z.setPredtoInf_j$. Esto implica que el antecedente de la propiedad en s_z sigue siendo cierto. El consecuente también es cierto ya que no se modifican las variables asociadas al nodo k y no se originan mensajes *INF* desde este mismo nodo, siendo $k \neq j$.

- $\pi_z \in \{dltINF_x(y,\ m)\colon \{\mathrm{x},\ \mathrm{y}\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Esta acción consiste en eliminar un mensaje *INF* del canal. Si $x = j$ e $y = i$, el antecedente podría hacerse falso. En cambio, cuando $y = k$, $k \neq j$ y $x = i$, el consecuente podría llegar a ser cierto.

- $\pi_z \in \{rcvINF_i(j,\ m)\colon\ m \in M_{INF}\}$. Al retirar el mensaje *INF* del canal siendo, el antecedente de la propiedad podría llegar a ser falso si, en s_{z-1}, era cierto únicamente por la existencia del mensaje *INF*.

- $\pi_z \in \{rcvINF_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Si $id = i$ y $(i,\ t) \in s_{z-1}.setPredToInf_j \setminus s_{z-1}.setPred_j$, la hipótesis inductiva asegura que el consecuente de la propiedad es cierto en s_{z-1}. Como $(i,\ t) \in s_{z-1}.setPredToInf_j$, la propiedad B.2.6 asegura que $\nexists\ m \in M_{INF}\colon m \in s_z.channel(j,\ i)$ y $m.ta = t$. Tras la ejecución de la acción en este supuesto, se forma un mensaje *INF*, $m\prime$, tal que $m\prime \in s_z.channel(j,\ i)$ y $m\prime.ta = t$, que hace que el antecedente siga siendo cierto en s_z. Así mismo, el consecuente es cierto en s_z porque las variables asociadas al nodo k de la propiedad no cambian y no se genera ningún mensaje *INF* desde ese nodo.

■

La siguiente propiedad indica que si un mensaje *ALG* va hacia un nodo *aborted*, su tiempo a la fuerza es incorrecto.

Propiedad B.2.8. Se verifica que $\forall\ i \in \mathcal{N}\ s.status.alg_i = aborted \wedge \exists m\colon (m.type = ALG \wedge m \in s.channel(j,\ i)) \Rightarrow m.ta < s.ta_i$.

Demostración: En el estado inicial, s_0, se verifica que $\forall\ i \in \mathcal{N}\colon\ s_0.status.alg_i = active$. Para confirmar que la propiedad es cierta hay que analizar los efectos que modifican el estado de un nodo, los que cambian su tiempo de activación y los que crean/eliminan mensajes *ALG*.

- $\pi_z = StartAddArc_i(j)$. Los efectos de la acción hacen que $s_z.status.alg_i \in \{blocked, candidate\}$. Por otro lado, puede crearse un mensaje ALG, $m \in s_z.channel(i, id)$, con $m.ta = t$. La propiedad B.2.2 establece que, para ese mensaje, $m.ta \leq s_z.ta_{id}$. Sólo en el caso de que $m.ta = s_z.ta_{id}$, la propiedad podría llegar a ser falsa. Sin embargo, aplicando la propiedad del medio B.1.4 se llega a la conclusión de que esto es posible si $s_z.state_i = blocked$ o, según la propiedad B.3.4, $s_z.status.alg_i \in \{blocked, dummy, candidate, victim\}$. Por tanto, el nuevo mensaje ALG cumple también la propiedad.

- $\pi_z = \{EndAddArc_i(j, t)\colon t \in \mathbb{N}\}$. Para que la acción esté habilitada $s_{z-1}.state_i \neq aborted$, o lo que es lo mismo, por la propiedad B.3.5, $s_{z-1}.status.alg_i \neq aborted$. Los efectos de la acción no modifican ta_i ni $status.alg_i$. Eso quiere decir que, tras la ejecución de la acción, la propiedad se cumple por hipótesis inductiva. Además, es posible que se generen mensajes ALG tal que $m \in s_z.channel(i,j)$ y $m.ta = t$. La propiedad B.2.2 indica que $1 \leq m.ta \leq s_z.ta_j$. En caso de que $m.ta = s_z.ta_j$, la propiedad podría ser falsa. Pero eso no es posible ya que los efectos de la acción hacen que $(j,t) \in s_z.setPred_i$. Según la propiedad B.3.2, $(j,t,received) \in s_z.set\_waiters_i$. Aplicando la propiedad del medio B.1.4, se llega a la conclusión de que o bien $s_z.state_j = blocked$ y $s_z.t\_activ_j = t$ o bien $s_z.state_j = aborted$ y $s_z.t\_activ_j > t$. Con la propiedad B.3.1 se tiene que $t = s_z.ta_j$ y en consecuencia $s_z.state_j = blocked$. La propiedad B.3.4 indica que entonces $s_z.status.alg_j \in \{blocked, dummy, candidate, victim\}$. Así que, la propiedad se verifica.

- $\pi_z = EndDelArc_i(j)$. Como los efectos de la acción hacen $s_z.status.alg_i = active$, el incremento del valor de ta_i no afecta en el cumplimiento de la propiedad.

- $\pi_z = Abort_i$. Tras la ejecución de la acción, $s_z.status.alg_i = aborted$. La propiedad podría ser falsa si en s_z existe un mensaje ALG que no cumple la condición temporal. Como la acción no genera ni elimina ningún mensaje de este tipo, ese mensaje ya tendría que existir en s_{z-1} cumpliendo que $m.ta \leq s_{z-1}.ta_i$ según la propiedad B.2.2. Los efectos de la acción modifican el tiempo de activación, de manera que $s_{z-1}.ta_i < s_z.ta_i$. Así que, es imposible que $m.ta$ permanezca constante al ejecutarse la acción y $m.ta < s_z.ta_i$. En consecuencia, si existe un mensaje AlG en s_z, éste verifica la propiedad.

- $\pi_z = initiate_j$. Al ejecutar esta acción, $s_z.status.alg_j = candidate$. Por otro lado, hay que comprobar que el mensaje ALG generado cumple también la propiedad. El nuevo mensaje va dirigido al nodo i cumple la propiedad B.2.2 y $m.ta \leq s_z.ta_i$. Como $(i,t) \in s_{z-1}.setPredecessors_j$, sabe por la propiedad B.3.2 que $(i, t, received) \in s_{z-1}.set\_waiters_j$. Considerando la propiedad del medio B.1.4, se plantean dos posibles situaciones: *($s_{z-1}.state_j = blocked$ y $s_{z-1}.t\_activ_i = t$*

o $s_{z-1}.state_i = aborted$ y $s_{z-1}.t\_activ_i > t$. La propiedad B.3.1 establece que $s_{z-1}.t\_activ_i = s_{z-1}.ta_i$, $m.ta = s_{z-1}.ta_i = t$. Los efectos de la acción no cambian ta_i ni $status.alg_i$. Por tanto, como $m.ta = s_z.ta_i = t$ y $t\_activ_i = t$, se descarta que $s_z.state_i = aborted$. Según la propiedad del medio B.1.4, $s_z.state_i = blocked$. Por todo ello, la propiedad se cumple.

- $\pi_z = \{rcvALG_i(j, m)\colon m \in M_{ALG}\}$. Al ejecutar esta acción se puede producir un cambio de estado, $s_z.status.alg_i \in \{dummy, victim\}$. La acción también puede generar un mensaje ALG que debe cumplir la propiedad.

  Si el consecuente es falso: $\exists\ m$: $(m.type = ALG \wedge m \in s_z.channel(i, y) \wedge m.ta = s_z.ta_y)$, para que la propiedad sea cierta $s_z.status.alg_x \neq aborted$. Como $(y, t_y) \in s_{z-1}.setPred_x$, aplicando la propiedad B.3.2 y la propiedad del medio B.1.4, se llega a la conclusión de que $(s_{z-1}.state_y = blocked \wedge s_{z-1}.t\_activ_y = t_y) \vee (s_{z-1}.state_y = aborted \wedge s_{z-1}.t\_activ_y > t_y)$. Considerando que $m.ta = s_z.ta_y \wedge s_z.ta_y = s_z.t\_activ_y = t_y$, sólo es posible que $s_{z-1}.state_y = blocked$. El envío del mensaje ALG va acompañado de un cambio de estado, así que $s_z.state_y = blocked \neq aborted$.

  Teniendo en cuenta que los efectos no cambian ta_y, $m.ta = s_{z-1}.ta_y$ y según la propiedad B.3.1, $s_{z-1}.t\_activ_y = s_{z-1}.ta_y = t_y$. Por lo tanto, la propiedad se cumple con el antecedente y el consecuente falso, $s_z.state_y = blocked \neq aborted \wedge m.ta = s_z.ta_y = t_y$.

- $\pi_z = \{rcvINF_i(j, m)\colon m \in M_{INF}\}$. Puede darse un cambio de estado que hace cierta la propiedad, $s_z.status.alg_i = candidate$. Según los efectos, también se puede crear un mensaje ALG. Si este mensaje verifica que $m.ta < s_z.ta_n$, entonces $s_z.status.alg_n \in \{blocked, dummy, candidate, victim\} \neq aborted$ (el mismo razonamiento que en las acciones *EndAddArc*, *Abort*, *initiate*, $rcvALG$).

- $\pi_z = \{rcvAVS_i(j, m)\colon m \in M_{AVS}\}$. En esta acción no se genera ni se elimina ningún mensaje ALG y no se modifica ta_i. En algún caso $status.alg_i$ podría cambiar su valor, pero $s_z.status.alg_i = victim$. En el resto de las situaciones, la propiedad se cumple por hipótesis inductiva.

■

Si un nodo i pasa a ser *aborted*, puede aparecer en los conjuntos *setPred* y *setPredToInf* de cualquier otro nodo, pero su tiempo asociado será inferior a su tiempo de activación actual, esto es, el tiempo registrado para el nodo i no estará actualizado.

Propiedad B.2.9. $\forall\ \{i, j\} \subseteq \mathcal{N}$ se verifica que $s.status.alg_i = aborted \wedge ((i,t) \in s.setPred_j \vee (i,t) \in s.setPredToInf_j) \Rightarrow 1 \leq t < s.ta_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$. Para comprobar que la propiedad se cumple en el estado s_z se deben analizar los efectos de las acciones que afectan a las variables involucradas.

- $\pi_z = StartAddArc_i(j)$. Tras la ejecución de la acción, $s_z.status.alg_i \in \{blocked, candidate\}$.

- $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$. Si la propiedad se cumple en s_{z-1} con $(i,t) \in s_{z-1}.setPred_j$ y $s_{z-1}.status.alg_i = aborted$ entonces, por hipótesis inductiva, $1 \leq t < s_{z-1}.ta_i$. Al ejecutarse la acción, el conjunto $setPred_j$, al igual que las variables ta_i y $status.alg_i$, no se modifican. Por consiguiente, la propiedad sigue siendo cierta en s_z.

- $\pi_z = \{EndAddArc_i(j,\ t)\colon t \in \mathbb{N}\}$. La condición de habilitación de la acción indica que $s_{z-1}.state_i \neq aborted$, o lo que es lo mismo según la propiedad B.3.5, $s_{z-1}.status.alg_i \neq aborted$. Si la propiedad se cumple en s_{z-1} por inducción, al ejecutarse la acción, la propiedad seguirá cumpliéndose en s_z porque la variable $status.alg_i$ no cambia.

- $\pi_z = \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$. Si $s_{z-1}.status.alg_i = aborted$ y se ejecuta la acción, el par (i,t) pasa a ser un elemento de $s_z.setPred_j$ y a veces de $s_z.setPredToInf_j$ (variables incluidas en la definición de $recordedWaits(s_z)$). Por tanto, la propiedad B.2.1 establece que $1 \leq t \leq s_z.ta_i$. Como $(i,t) \in s_z.setPred_j$, la propiedad B.3.2 indica que $(i,t,received) \in s_z.set\_waiters_j$. Entonces aplicando la propiedad B.1.4, se llega a la conclusión de que $s_z.t\_activ_i > t$ porque $s_z.status.alg_i = aborted$. Por otra parte, la propiedad B.3.1 señala que $s_z.t\_activ_i = s_z.ta_i$. Esto implica que $s_z.ta_i > t$ y, en consecuencia, $1 \leq t < s_z.ta_i$.

- $\pi_z = \{StartDelArc_i(j,\ t)\colon t \in \mathbb{N}\}$. Si $s_{z-1}.status.alg_i = active$, el estado del nodo i no cambia al ejecutarse la acción y la hipótesis inductiva concluye. Cuando $s_{z-1}.status.alg_j = aborted$, la hipótesis inductiva permite demostrar la propiedad en s_z. Los efectos de la acción no modifican ni ta_i ni $setPred_j$ ni $setPredtoInf_j$ y, además, el nodo i mantiene su estado.

- $\pi_z = \{StartDelArc_j(i,\ t)\colon t \in \mathbb{N}\}$. Cuando $s_{z-1}.status.alg_i = aborted$, como $(i,t) \in s_{z-1}.setPred_j$, la hipótesis inductiva asegura que $1 \leq t < s_{z-1}.ta_i$. Aunque el par (i,t) deja de ser un elemento de $setPred_j$ y de $setPredToInf_j$, los efectos de la acción no cambian ni ta_i ni $status.alg_i$. Por consiguiente, la propiedad sigue cumpliéndose en s_z.

- $\pi_z = \{EndDelArc_i(j)\}$. Aunque los efectos de la acción incrementan ta_i de modo que $s_{z-1}.ta_i < s_z.ta_i$, también $s_z.status.alg_i = active$. El nuevo valor que adquiere

la variable $status.alg_i$ es suficiente para asegurar que la propiedad es cierta en s_z.

- $\pi_z = Abort_i$. Al ejecutarse la acción, $s_z.status.alg_i = aborted$ y ta_i se incrementa en una unidad ($s_{z-1}.ta_i < s_z.ta_i$). Si $(i,\ t)$ era un elemento de $setPred_j$ o $setPredToInf_j$ en s_{z-1}, la propiedad B.2.1 confirma que $1 \leq t \leq s_{z-1}.ta_i$. La única posibilidad de que no se cumpla la condición temporal consiste en que $1 \leq t = s_{z-1}.ta_i$, pero el cambio de ta_i hace evidente que la propiedad se cumple en s_z porque $1 \leq t < s_z.ta_i$.

- $\pi_z = Abort_j$. Los efectos de la acción vacían tanto el conjunto $s_z.setPred_j$ como el conjunto $s_z.setPredToInf_j$.

- $\pi_z = initiate_i$. Al ejecutarse la acción, $s_z.status.alg_i = candidate$.

- $\pi_z = initiate_j$. Si se cumple la propiedad en s_{z-1}, sabiendo que $(i,\ t) \in s_{z-1}.setPred_j$, $1 \leq t < s_{z-1}.ta_i$. Al ejecutarse la acción en este supuesto, no se modifican las variables implicadas: $status.alg_i$, ta_i y $setPred_j$. Por tanto, se verifica la propiedad por hipótesis inductiva.

- $\pi_z \in \{rcvALG_i(j,\ m)\colon m \in M_{ALG}\}$. Tras la ejecución de la acción, $s_z.status.alg_i \in \{dummy,\ candidate,\ victim\}$.

- $\pi_z \in \{rcvALG_j(y,\ m)\colon y \in \mathcal{N} \wedge m \in M_{ALG}\}$. Si $s_{z-1}.status.alg_i = aborted$ y $(i,\ t) \in s_{z-1}.setPred_j$, se asume por hipótesis inductiva que $1 \leq t < s_{z-1}.ta_i$. Los efectos de la acción en este supuesto mantienen todos los valores de las variables involucradas en la propiedad. Por tanto, la propiedad se cumple. Para cualquier otro caso, la hipótesis inductiva concluye.

- $\pi_z = \{rcvINF_i(j,\ m)\colon m \in M_{INF}\}$. Los efectos de la acción pueden modificar $status.alg_i$ de manera que $s_z.status.alg_i = candidate$. Cuando $status.alg_i$ mantiene su valor al ejecutar la acción, la hipótesis inductiva concluye.

- $\pi_z = \{rcvAVS_i(j,\ m)\colon m \in M_{AVS}\}$. Al ejecutarse esta acción, si $s_{z-1}.status.alg_i = candidate$, los efectos podrían hacer que $s_z.status.alg_i = victim$. En el resto de casos, la propiedad se demuestra aplicando la hpótesis inductiva.

■

La siguiente propiedad confirma que, cuando un nodo pasa a ser *aborted*, se eliminan todos los elementos que pudieran contener los conjuntos *setPred* y *setPredToInf* asociados a él.

Propiedad B.2.10. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = aborted \Rightarrow s.setPred_i = s.setPredToInf_i = \emptyset$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$. Seguidamente se analizan los efectos de las acciones que influyen en las variables de la propiedad.

- $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$ o $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ o $\pi_z = initiate_i$ o $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ o $\pi_z \in \{rcvAVS_i(x, m)\colon x \in \mathcal{N}\} \wedge m \in M_{AVS}\}$. Al ejecutarse cualquiera de estas acciones, $status.alg_i$ puede cambiar de tal forma que $s_z.status.alg_i \in \{active, blocked, dummy, candidate, victim\}$. Si no se produce el cambio de estado mencionado, la hipótesis inductiva concluye.

- $\pi_z \in \{EndAddArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Para que la acción esté habilitada es preciso que $s_{z-1}.state_i \neq aborted$ o, aplicando la propiedad B.3.5, $s_{z-1}.status.alg_i \neq aborted$. Por otra parte, los efectos de la acción no modifican la variable $status.alg$. Así que, se concluye que la propiedad se cumple en s_z.

- $\pi_z \in \{StartDelArc_i(j, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Si $s_{z-1}.status.alg_i = active$, la hipótesis inductiva concluye. En el resto de estados posibles del nodo i, la propiedad también se verifica aplicando la hipótesis inductiva. Hay que recordar que la acción no está habilitada si $s_{z-1}.status.alg_i = aborted$. En ese estado, la propiedad B.1.3 establece que $(j, t, sent) \in s_{z-1}.set\_waiters_i$ y la acción requiere que sea la tupla $(j, t, received)$ la que pertenezca a $set\_waiters_i$ en s_{z-1}.

- $\pi_z = Abort_i$. Al ejecutarse la acción, es obvio que la propiedad se verifica porque $s_z.status.alg_i = aborted$, $s_z.setPred_i = \emptyset$ y $s_z.setPredToInf_i = \emptyset$.

■

Esta propiedad asegura que los conjuntos *setPred* y *setPredToInf* asociados al nodo i contienen los mismos elementos si el estado del nodo i es *dummy* o *candidate*.

Propiedad B.2.11. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i \in \{dummy, candidate\} \Rightarrow s.setPred_i = s.setPredToInf_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$. Para demostrar que la propiedad se cumple en s_z hay que analizar los efectos de las acciones que modifican las variables $status.alg_i$, $setPred_i$ y $setPredToInf_i$.

- $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$. Si los efectos de la acción hacen que $s_z.status.alg_i = blocked$, es evidente que la propiedad es cierta. Otro efecto posible de la acción

modifica el estado del nodo i de manera que $s_z.status.alg_i = candidate$. En ese caso, el cambio de estado viene acompañado por otro efecto que iguala los conjuntos $setPred_i$ y $setPredToInf_i$. En consecuencia, la propiedad se cumple.

- $\pi_z = \{EndAddArc_i(j, t)\colon t \in \mathbb{N}\}$. Cuando $s_{z-1}.status.alg_i \in \{dummy, candidate\}$ se modifica el conjunto $setPredToInf_i$ añadiendo el elemento (j, t). Considerando la hipótesis inductiva y teniendo en cuenta que ese mismo elemento se incorpora previamente a $setPred_i$ por efecto de la misma acción, se concluye que la propiedad se cumple porque los conjuntos quedan igualados.

- $\pi_z = \{StartDelArc_i(j, t)\colon t \in \mathbb{N}\}$. Si $s_{z-1}.status.alg_i \in \{dummy, candidate\}$, los efectos de la acción no modifican el estado del nodo i, pero eliminan el par (j, t) tanto de $setPred_i$ como de $setPredToInf_i$. Como en s_{z-1} se cumplía la propiedad por hipótesis inductiva, después de retirarse (j, t) de ambos conjuntos la propiedad sigue verificándose. Los efectos que se producen sobre $setPred_i$ y $setPredToInf_i$ cuando $s_{z-1}.status.alg_i = active$ no afectan al cumplimiento de la propiedad porque el estado del nodo i no cambia en ese caso.

- $\pi_z = \{EndDelArc_i(j)\}$ o $\pi_z = Abort_i$. Tras la ejecución de estas acciones, $s_z.status.alg_i \in \{active, aborted\}$.

- $\pi_z = initiate_i$. La ejecución de la acción trae consigo los siguientes efectos: $s_z.status.alg_i = candidate$ y $s_z.setPredToInf_i = s_z.setPred_i$. Por tanto, la propiedad se verifica.

- $\pi_z = \{rcvALG_i(j, m)\colon m \in M_{ALG}\}$. Al ejecutarse la acción, $status.alg_i$ puede convertirse en *victim*, pasar a ser *dummy* o mantener su estado como *candidate*. En la primera situación es obvio que la propiedad se cumple. En el caso de que $s_z.status.alg_i = dummy$, seguidamente otro efecto hace que se igualen los conjuntos, $s_z.setPred_i = s_z.setPredToInf_i$. Por último, si $s_z.status.alg_i = candidate$, la hipótesis inductiva confirma que $s_z.setPred_i$ y $s_z.setPredToInf_i$ tienen los mismos elementos porque en el estado previo el nodo i era *candidate* y los conjuntos mencionados no varían al ejecutarse este caso.

- $\pi_z = \{rcvINF_i(j, m)\colon m \in M_{INF}\}$. En caso de que los efectos de la acción no modifiquen $status.alg_i$, la propiedad se demuestra aplicando la hipótesis inductiva ($s_{z-1}.status.alg_i \in \{active, blocked\}$). Si, al ejecutarse la acción, resulta que $s_z.status.alg_i = candidate$, el cambio de estado del nodo i viene acompañado de la modificación de $setPredToInf_i$ de tal forma que su contenido se iguala con el de $setPred_i$. Así que, en este caso la propiedad también se verifica .

- $\pi_z = \{rcvAVS_i(j,\ m)\colon m \in M_{AVS}\}$. Entre los posibles efectos de la acción se encuentra el que hace que $s_z.status.alg_i = victim$. En el resto de casos, la hipótesis inductiva permite concluir.

■

De acuerdo con la siguiente propiedad, todo nodo que es seleccionado como víctima o pasa a ser *aborted* deja de guardar información en la variable asociada st_alg_i. Con ello se consigue que los nodos en estos estados no difundan información que va a pasar a ser falsa.

Propiedad B.2.12. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i \in \{victim,\ aborted\} \Rightarrow s.st\_alg_i = NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ y $s_0.st\_alg_i = NULL$ por lo que la propiedad es cierta. A continuación se repasan los efectos de las acciones que influyen en las variables aludidas por la propiedad.

Las únicas acciones que pueden convertir el estado del nodo i en *victim* son $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$. En ambos casos se produce otro efecto que conlleva que $s_z.st\_alg_i = NULL$. La propiedad, por tanto, se verifica. Por otro lado, la acción $\pi_z = Abort_i$ transforma $status.alg_i$ en *aborted*, pero tiene como precondición que $s_{z-1}.status.alg_i = victim$. Aplicando la hipótesis inductiva, se deduce que $s_{z-1}.st\_alg_i = NULL$. Como la acción no modifica la variable st_alg_i, se concluye que en s_z la propiedad se sigue cumpliendo.

En el resto de acciones no se puede hacer cierto el antecedente ni falso el consecuente por lo que la aplicación de la hipótesis inductiva basta para demostrar la propiedad.

■

Esta propiedad confirma que cualquier nodo en estado *blocked* o *active*, que tenga conocimiento pleno de su identidad ($status.id = known$), no tiene información almacenada en su correspondiente st_alg.

Propiedad B.2.13. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i \in \{blocked,\ active\} \wedge s.status.id_i = known \Rightarrow s.st\_alg_i = NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ por lo que la propiedad es cierta.

Las acciones del conjunto $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ tienen como condición de habilitación que $s_{z-1}.state_i = active$ o, por la propiedad B.3.3, $s_{z-1}.status.alg_i = active$. En el caso de que $s_{z-1}.status.id_i = unknown$, la propiedad se verifica porque,

tras la ejecución de la acción, no se producen cambios en la variable $status.id_i$. Sin embargo, cuando $s_{z-1}.status.id_i = known$, la hipótesis inductiva establece que en $s_{z-1}.st\_alg_i$ no se guarda ninguna información. Si $s_z.status.alg_i$ se convierte en *blocked* o *candidate*, la propiedad se cumple porque al ejecutarse la acción no se producen cambios ni en $status.id_i$ ni en st_alg_i.

Por otro lado, la ejecución de la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ hace que $s_z.status.alg_i = active$. Si este efecto conlleva otro que cancele el contenido de st_alg_i, es obvio que la propiedad se cumple. En el resto de casos, el cambio de estado va acompañado del efecto que transforma $s_z.status.id_i$ en *unknown*, haciendo falso el antecedente en s_z.

Los cambios de estado que resultan al ejecutar las acciones $\pi_z = Abort_i$ ($s_z.status.alg_i = aborted$), $\pi_z = initiate_i$ ($s_z.status.alg_i = candidate$), $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ ($s_z.status.alg_i \in \{dummy,\ victim\}$), $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ ($s_z.status.alg_i = victim$) y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ ($s_z.status.alg_i = candidate$) hacen el antecedente falso y, por tanto, la propiedad se cumple en s_z.

La acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ es la única que cambia el valor de $status.id_i$ a *known*. Como seguido de este efecto se vacía st_alg_i, resulta evidente que la propiedad también es cierta después de su ejecución.

Examinando los posibles cambios en st_alg_i, sólo queda comentar el efecto de la acción $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ que consiste en $s_z.st\_alg_i = NULL$. Obviamente, la propiedad también se verifica en este caso.

■

Esta propiedad asegura que sólo los nodos en estado *candidate* o *victim* pueden guardar información en la variable *st_avsrsp* asociada.

Propiedad B.2.14. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i \notin \{victim,\ candidate\} \Rightarrow s.st\_avsrsp_i = NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_avsrsp_i = NULL$ por lo que la propiedad es cierta.

Entre las acciones que pueden modificar $status.alg_i$ a estados no admitidos en la propiedad, se encuentran $\pi_z = initiate_i$ ($s_z.status.alg_i = candidate$), $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ ($s_z.status.alg_i = victim$), $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ ($s_z.status.alg_i = victim$) y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ ($s_z.status.alg_i = candidate$). En estos casos, la propiedad se cumple porque el antecedente de la implicación es falso. También hay que analizar las acciones cuyos efectos cambian el valor de $status.alg_i$, pero se alcanza un estado permitido.

La acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ está habilitada cuando $s_{z-1}.status.alg_i = active$ (equivalente a $s_{z-1}.state_i = active$ por la propiedad B.3.3). La hipótesis inductiva establece que $s_{z-1}.st\_avsrsp_i = NULL$. Tras la ejecución de la acción,

$s_z.status.alg_i$ puede ser *blocked*, pero st_avsrsp_i sigue sin contener información porque los efectos no lo modifican. De igual forma, aplicando la hipótesis inductiva en la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ cuando $s_{z-1}.status.alg_i = dummy$ o la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ si $s_{z-1}.status.alg_i = blocked$, se asume que $s_{z-1}.st\_avsrsp_i = NULL$. Aunque al ejecutarse las acciones $s_z.status.alg_i \in \{active,\ dummy\}$, st_avsrsp_i se mantiene sin información. Por consiguiente, la propiedad también se verifica. Al ejecutarse la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ cuando $s_{z-1}.status.alg_i = candidate$ o $\pi_z = Abort_i$, se cancela el contenido de $s_z.st\_avsrsp_i$. Este efecto basta para comprobar que la propiedad es cierta en s_z.

Por el contrario, las acciones $\pi_z \in \{rcvAVSRSP_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$ y $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ sólo si $s_{z-1}.status.alg_i = candidate$, almacenan el mensaje que retiran del canal en $s_z.st\_avsrsp_i$. Como este cambio en la variable st_avsrsp_i no viene acompañado de un cambio de estado, se confirma que la propiedad se cumple.

■

Según la siguiente propiedad, los únicos estados en los que un nodo puede permanecer sin conocer su identidad (simulada), $s.status.id_i = unknown$, son *active* y *blocked*. Estos son los estados en los que el nodo bloquea las instancias del algoritmo.

Propiedad B.2.15. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.id_i = unknown \Rightarrow s.status.alg_i \in \{active,\ blocked\}$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.id_i = known$ por lo que la propiedad es cierta. Para comprobar que tras la ejecución de las acciones la propiedad sigue cumpliéndose, se van a analizar sólo los efectos que modifican las variables *status.id* y *status.alg*.

Las acciones $\pi_z = initiate_i$ o $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ modifican la variable $status.alg_i$, pero para que estén habilitadas, precisan que $s_{z-1}.status.id_i$ sea *known*. Como los efectos de estas acciones no alteran el valor $status.id_i$, es evidente que la propiedad se cumple. Del mismo modo, la acción $\pi_z = Abort_i$ tiene como precondición $s_{z-1}.status.alg_i = victim$. Considerando la hipóstesis inductiva se sabe que en ese estado $s_{z-1}.status.id_i = known$. Puesto que el valor de esta variable no cambia al ejecutarse la acción, se concluye. Por otra parte, la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ tiene como efecto $s_z.status.id_i = known$. Al adquirir la variable este valor, la propiedad queda demostrada.

En lo que respecta a la variable *status.alg*, se puede apreciar que la ejecución de la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ hace que el estado del nodo i pase a ser *active* y la propiedad se verifique. Esta misma acción es la única que cambia el valor de la

variable $s_z.status.id_i$ a *unknown*. Como en ese caso $s_z.status.alg_i$ es *active*, el efecto no afecta al cumplimiento de la propiedad.

Así mismo, la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ puede convertir el estado del nodo i en *blocked* que es otro de los estados permitidos en el consecuente de la propiedad. Otro efecto posible de la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ consiste en que $s_z.status.alg_i = candidate$. La propiedad también se verifica porque ese cambio sólo se produce si $s_{z-1}.status.id_i = known$ y ese valor se mantiene tras la ejecución de la acción.

■

La siguiente propiedad demuestra que las rutas, que almacena un nodo en *set_st_avs* y que llegaron mediante mensajes *AVS*, han recogido información de esperas a ese nodo, que no se han borrado aún.

Propiedad B.2.16. Si $\exists\ i \in \mathcal{N},\ t \in \mathbb{N},\ sid \in T,\ path \in P^+,\ b \in \{true, false\}$ verificando que $(sid, t, path, b) \in s.set\_st\_avs_i \Rightarrow penult(path) \in s.setPred_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.set\_st\_avs_i = \emptyset$, por lo que la propiedad se cumple. Seguidamente se van a analizar los efectos de aquellas acciones que actúan sobre las variables aludidas en la propiedad.

En lo que respecta al conjunto $setPred_i$, la acción $\pi_z \in \{EndAddArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ se encarga de añadir nuevos elementos, pero sus efectos no cambian el contenido de $s_{z-1}.set\_st\_avs_i$. Por tanto, la hipótesis inductiva permite verificar la propiedad. La retirada de elementos de $setPred_i$ se produce al ejecutarse $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Aunque el par (j, t) que se elimina fuera el penúltimo elemento de la ruta de algún mensaje almacenado en $s_{z-1}.set\_st\_avs_i$, dicho mensaje, tras la ejecución de la acción, desaparecería de $set\_st\_avs_i$.

Al ejecutarse la acción $\pi_z = Abort_i$ se ven afectadas las dos variables de la propiedad. Los efectos de esta acción vacían tanto $set\_st\_avs_i$ como $setPred_i$. Es obvio que la propiedad se verifica en s_z.

Las acciones que modifican la variable $set\_st\_avs_i$ son: $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$, $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, $\pi_z = sndAVS_i$ y $\pi_z = sndAVSRSP_i$. La ejecución de la primera puede almacenar el mensaje *AVS*, m, en $s_z.set\_st\_avs_i$ sólo si $penult(m.path) \in s_{z-1}.setPred_i$. Al cumplirse esta condición, se confirma que se verifica la propiedad en s_z porque $setPred_i$ no varía.

Por otro lado, las acciones $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z = sndAVS_i$ eliminan uno de los mensajes almacenados en $s_{z-1}.set\_st\_avs_i$ haciendo falso el antecedente de la implicación. Para el resto de mensajes de este almacén basta con aplicar la hipótesis inductiva para afirmar que la propiedad se cumple. La última acción modifica el contenido de un mensaje de $s_{z-1}.set\_st\_avs_i$, pero el cambio no

afecta a la ruta, *path*, del mismo. Por lo tanto, la propiedad queda demostrada en este caso considerando la hipótesis inductiva. ■

La variable *cand_succ* recoge información sobre el candidato por el que espera transitivamente un nodo. Esto permite que ambos candidatos colaboren en la fase de eliminación de candidatos. Obviamente, esto tiene sentido sólo si el nodo que conoce su candidato sucesor es también candidato.

Propiedad B.2.17. $\forall\ i \in \mathcal{N}$ se verifica que $s.cand\_succ_i \neq NULL \Rightarrow s.status.alg_i = candidate$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.cand\_succ_i = NULL$ por lo que la propiedad es cierta.

Considerando la variable $cand\_succ_i$, tanto la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ como las acciones $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ pueden anular su valor y, por tanto, hacer que el antecedente de la propiedad sea falso en s_z. Cuando en $cand\_succ_i$ se introduce un nuevo valor por efecto de las acciones $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVSRSP_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$, el estado del nodo i es *candidate*. Así que, en esa situación la propiedad se verifica.

Los cambios en la variable $status.alg_i$ que hacen adopte el valor *candidate* se producen al ejecutar: $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z = initiate_i$ o $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. La propiedad es este caso se cumple con el consecuente cierto. A diferencia de estas acciones, los efectos de $\pi_z = Abort_i$ y $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ pueden modificar $status.alg_i$ a valores distintos a *candidate*, esto es, $s_z.status.alg_i = aborted$ y $s_z.status.alg_i = dummy$ respectivamente. En ambos casos, la propiedad se verifica en s_{z-1} siendo $s_{z-1}.status.alg_i \neq candidate$ y la hipótesis inductiva concluye porque el valor de $cand\_succ_i$ no varía.

■

En esta propiedad se establece que el destino de un mensaje *AVSRSP* y el tiempo que sirve para validarlo coincide con el par (identificador, tiempo) del primer elemento de la ruta que contiene el mensaje.

Propiedad B.2.18. Si $\exists\ \{i, j\} \subseteq \mathcal{N}$, $t_j \in \mathbb{N}$, $sid \in T$, $path \in P^+$ verificando $(AVSRSP, t_j, sid, path) \in s.channel(i, j) \Rightarrow first(path) = (j, t_j)$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \subseteq \mathcal{N}$ se cumple que $s_0.channel(j, i) = \epsilon$ por lo que la propiedad es cierta. Las acciones que generan mensajes *AVSRSP* son: $\pi_z \in \{rcvAVS_i(j, m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$, $\pi_z \in \{rcvINF_i(j, m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{INF}\}$ y $\pi_z = sndAVSRSP_i$. Las dos primeras acciones transforman el mensaje

AVS, $m\prime$, recibido o almacenado respectivamente, en un mensaje *AVSRSP*, m, tal que *first*$(m.path)$ = *first*$(m\prime.path)$. Además, $m.ta$ = *first*$(m\prime.path).ta$ y el destino del mensaje es *first*$(m\prime.path).id$. Por último, la ejecución de $\pi_z = sndAVSRSP_i$ forma un mensaje m que tiene la siguiente forma: (*AVSRSP*, *first*$(path).ta$, $s_z.st\_avsrsp_i.sid$, $path - last(path).s_z.st\_avsrsp_i.path$). Como se puede observar *first*$(m.path)$ coincide con *first*$(path)$. Esto hace evidente que $m.ta$ = *first*$(path).ta$ y el mensaje va dirigido al nodo *first*$(path).id$.

■

De manera semejante a la propiedad anterior, se señala que el primer elemento de una ruta almacenada en la variable *st_avsrsp* de un nodo es precisamente el par formado por el identificador de ese nodo y por su correspondiente tiempo de activación actualizado.

Propiedad B.2.19. Si $\exists\ i \in \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $path \in P^+$ verificando $(t, sid, path) \in s.st\_avsrsp_i \Rightarrow$ *first*$(path) = (i,\ s.ta_i)$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_avsrsp_i = NULL$ por lo que la propiedad es cierta.

Considerando los cambios de las variables incluidas en la propiedad al ejecutarse la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$, resulta que el tiempo de activación del nodo i se incrementa de manera que $s_{z-1}.ta_i < s_z.ta_i$. Este efecto no influye en el cumplimiento de la propiedad porque el nodo i alcanza el estado *active* en s_z. De acuerdo con la propiedad B.2.14 en ese estado $s_z.st\_avsrsp_i$ no guarda ninguna información. Del mismo modo, la ejecución de la acción $\pi_z = Abort_i$ hace que $s_z.st\_avsrsp_i$ sea *NULL* y, por tanto, la propiedad cierta.

Tanto la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ como la acción $\pi_z \in \{rcvAVSRSP_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ tienen como efecto incluir un mensaje en st_avsrsp_i. El mensaje almacenado se forma a partir del mensaje *ALG* o *AVSRSP*, m, que se retira del canal asociado al nodo i. Según los efectos de la primera acción *first*$(s_z.st\_avsrsp_i.path) = (i,\ s_z.ta_i)$. Para la segunda acción, basta con aplicar la propiedad B.2.18 para asegurar que el mensaje *AVSRSP* verifica en s_{z-1} que *first*$(m.path)$ $= (i,t_i)$ y $m.ta = t_i$. Como el requisito para que el mensaje *AVSRSP* pase a almacenarse en st_avsrsp_i es que $m.ta = s_z.ta_i$, la propiedad queda demostrada.

■

En esta propiedad se asegura que los nodos candidatos con información en su correspondiente *st_avsrsp* conocen la identidad de su candidato sucesor, ya sea porque la han recibido de un mensaje *ALG* procedente de ese candidato, o porque la han extraído del campo *sid* de un mensaje *AVSRSP* que se ha almacenado.

Propiedad B.2.20. $\forall\ i \in \mathcal{N}$ se verifica que $s.st\_avsrsp_i \neq NULL \wedge s.status.alg_i = candidate \Rightarrow s.cand\_succ_i = s.st\_avsrsp_i.sid$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_avsrsp_i = NULL$, lo que implica que la propiedad es cierta.

La variable $cand\_succ_i$ puede verse modificada tras la ejecución de las acciones: $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$. En ambos casos, cuando $status.alg_i$ pasa a ser $victim$, $cand\_succ_i$ se cancela. La combinación de estos efectos permite demostrar la propiedad.

Los efectos de la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ pueden provocar que tanto $s_z.st\_avsrsp_i$ como $s_z.cand\_succ_i$ se anulen. Por tanto, en este caso, la propiedad también se cumple. En cambio, si se ejecuta la acción $\pi_z = Abort_i$, sólo cambia la variable $s_z.st\_avsrsp_i$. El antecedente de la propiedad es falso ya que $s_z.st\_avsrsp_i = NULL$.

En lo que respecta a la variable $status.alg_i$, ésta puede adquirir el valor $candidate$ al ejecutarse las acciones $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z = initiate_i$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. Como este cambio de estado se produce siendo $s_{z-1}.status.alg_i \notin \{candidate,\ victim\}$, la propiedad B.2.14 señala que $s_{z-1}.st\_avsrsp_i = NULL$. La propiedad queda probada debido a que la variable st_avsrsp_i no varía tras la ejecución de las acciones.

Por otra parte, los efectos de las acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z = \{rcvAVSRSP_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$ pueden almacenar un mensaje en $s_z.st\_avsrsp_i$, siendo $s_z.status.alg_i = candidate$. Además, en esos casos $s_z.cand\_succ_i$ pasa a contener $m.sid$. Como $m.sid$ coincide con $s_z.st\_avsrsp_i.sid$, la propiedad se verifica.

■

Esta propiedad indica que el último par (nodo, tiempo) de la ruta contenida, tanto en los mensajes AVS como en los mensajes almacenados en set_st_avs, coincide con el identificador del destino y el tiempo de esos mensajes o coincide con el nodo al está asociada la variable set_st_avs junto con el tiempo almacenado en ella.

Propiedad B.2.21. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$, $t_j \in \mathbb{N}$, $sid \in T$, $path \in P^+$, $b \in \{true,\ false\}$ verificando que $(AVS,\ t_i,\ sid,\ path,\ b) \in s.channel(j,\ i) \vee (sid,\ t_i,\ path,\ b) \in s.set\_st\_avs_i \Rightarrow last(path) = (i,\ t_i)$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \in \mathcal{N}$ se cumple que $s_0.channel(i,\ j) = \epsilon$ y $s_0.set\_st\_avs_i = \emptyset$. Por consiguiente, la propiedad es cierta.

En primer lugar, se van a considerar las acciones que pueden generar mensajes AVS. Las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvAVS_i(j,\ m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ construyen el mensaje AVS a partir de mensajes $m\prime$ almacenados en $s_{z-1}.set\_st\_avs_i$ o retirados del canal directamente (última

acción). La ruta de los nuevos mensajes *AVS* consiste en $m\prime.path - last(m\prime.path)$. Esto implica que $last(m.path) = penult(m\prime.path)$. Tal y como señala la propiedad, el destino del mensaje es $last(m.path).id$ y el tiempo asociado al mensaje es $last(m.path).ta$.

El mensaje *AVS* que se crea al ejecutar la acción $\pi_z = \{firstAVS_i\}$ va dirigido al nodo $last(st\_alg_i.path).id$ y el tiempo de ese nodo es $last(st\_alg_i.path).ta$. Como el último elemento de la ruta del nuevo mensaje es $last(st\_alg_i.path)$, la propiedad se verifica. Finalmente, la acción $\pi_z = sndAVS_i$ forma un mensaje *AVS* en el que su último elemento coincide con $last(st\_avsrsp_i.path)$. Dado que el destino del mensaje *AVS* y su tiempo se extraen de $last(st\_avsrsp_i.path)$, la propiedad queda probada.

En lo que respecta a la variable $set\_st\_avs_i$, las acciones $\pi_z \in \{StartDelArc_i(x, t)$: $x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = sndAVS_i$ y $\pi_z = Abort_i$ tan sólo la modifican eliminando mensajes. Este efecto hace el antecedente de la propiedad falso. Así mismo, al ejecutarse la acción $\pi_z = Abort_i$, $set\_st\_avs_i$ se vacía y, por tanto, también se cumple la propiedad.

A diferencia de estas acciones, la acción $\pi_z \in \{rcvAVS_i(x, m)$: $x \in \mathcal{N} \wedge m \in M_{AVS}\}$ incluye un mensaje en $set\_st\_avs_i$. Hay que comprobar que ese nuevo mensaje hace que la propiedad siga verificándose. El mensaje que se almacena procede de un mensaje *AVS* que estaba en el canal en el estado previo. En este caso la hipótesis inductiva concluye. ■

En esta propiedad se asegura que el campo *sid* de un mensaje *AVS* o de un mensaje almacenado en *set_st_avs* contiene la identidad simulada del primer nodo de la ruta de esos mensajes. En otras palabras, el campo *sid* permite dar a conocer, al último nodo de la ruta de los mensajes, la identidad simulada del candidato mayor que le precede en la ruta dada. .

Propiedad B.2.22. Si $\exists \{i, j, x\} \subseteq \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $path \in P^+$, $b \in \{true, false\}$ verificando que: $(AVS, t, sid, (x, s.ta_x)\cdot path, b) \in s.channel(j, i) \vee (sid, t, (x, s.ta_x)\cdot path, b) \in s.set\_st\_avs_i \Rightarrow sid = s.sim\_id_x$.

Demostración: En el estado inicial, s_0, $\forall \{i,j\} \in \mathcal{N}$ se cumple que $s_0.channel(j, i) = \epsilon \wedge s_0.set\_st\_avs_i = \emptyset$ por lo que la propiedad es cierta.

El efecto de las acciones de retirar un mensaje *AVS* del canal o de eliminar un mensaje almacenado de *set_st_avs* no es necesario analizarlo porque hace evidente el cumplimiento de la propiedad al hacer falso el antecedente de la misma.

Las acciones $\pi_z \in \{StartDelArc_i(j, t)$: $j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvAVS_i(j, m\prime)$: $j \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ generan mensajes *AVS*, *m*, a partir de mensajes almacenados en $s_{z-1}.set\_st\_avs_i$ (para las dos primeras acciones) o del mensaje $m\prime$ que se retiran del canal de comunicaciones (última acción). En los nuevos mensajes se copia el campo *sid* y la ruta, a excepción del último par, esto es, $m.sid = m\prime.sid$ y $m.path = m\prime.path - last(m\prime.path)$. Dado que los efectos de las acciones

$\{StartDelArc_i(j,\ t)\}$ y $\{rcvAVS_i(j,m)\}$ no alteran las variables incluidas en la propiedad y se preserva el primer elemento de la ruta existente en s_{z-1}, $first(m.path) = first(m\prime.path)$, la hipótesis inductiva concluye. Los mensajes AVS que se crean al ejecutar $\pi_z = Abort_i$ también verifican la propiedad ya que otro efecto de la acción consistente en aumentar el tiempo de activación del nodo i hace que el antecedente de la propiedad siempre sea falso.

Por lo contrario, los efectos de las acciones $\pi_z = \{firstAVS_i\}$ y $\pi_z = sndAVS_i$ consisten en la formción de un nuevo mensaje AVS, m. El mensaje resultante de la ejecución de la primera acción verifica que el primer elemento de su ruta es $(i,\ s_z.ta_i)$ y el campo sid del mismo es $s_z.sim\_id_i$. Es obvio entonces que la propiedad se cumple con el antecedente y consecuente ciertos porque $first(m.path) = (i,\ s_z.ta_i)$. La segunda acción crea un mensaje cuya primera parte de la ruta se corresponde con un mensaje almacenado en $s_{z-1}.set\_st\_avs_i$, $m\prime$. El nuevo mensaje cumple que $m.sid = m\prime.sid$ y $first(m.path) = first(m\prime.path)$. Como los efectos de la acción no modifican las variables ta ni sim_id de ningún nodo, y en particular del primer nodo de la ruta de $m\prime$, la hipótesis inductiva basta para comprobar que el nuevo mensaje verifica la propiedad.

Del mismo modo, las acciones $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z = sndAVSRSP_i$ pueden introducir nuevos mensajes en $set\_st\_avs_i$. En el primer caso, un mensaje AVS existente en s_{z-1} (cumple la propiedad por hipótesis inductiva) se retira del canal y se almacena sin ninguna modificación en el campo sid ni en los elementos de la ruta. En el segundo caso, el mensaje que se incorpora en $s_z.set\_st\_avs_i$ es idéntico a uno que estaba almacenado ya en s_{z-1} salvo en su último campo. Este campo no influye en la comprobación de la propiedad porque no se hace referencia a él. Por consiguiente, en ambos casos la propiedad se verifica ya que no se producen cambios en la variables ta y sim_id de ningún nodo.

En relación con la variable sim_id, sólo las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden alterar su valor. Para que sim_id_i adopte un nuevo valor por efecto de la ejecución de $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \in t \in \mathbb{N}\}$ se requiere que $s_{z-1}.status.alg_i = active$. Ese estado se adquiere únicamente al ejecutarse cualquier acción $EndDelArc_i(j)$. Los efectos de esta acción también incluyen un incremento en el tiempo de activación del nodo i. Así que, la modificación de sim_id_i en s_z que puede hacer que el consecuente de la propiedad sea falso lleva implícito un cambio en ta_i por el que el antecedente también es falso. Básicamente por la misma razón, cualquiera de los cambios de la variable sim_id_i que pueden tener lugar al ejecutarse la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ precisan que $s_{z-1}.status.id_i = unknown$. Ese estado sólo es alcanzable tras la ejecución previa de $EndDelArc_i(j)$. Al igual que en el caso anterior, el tiempo de activación del nodo i se habrá visto incrementado haciendo que para cualquier mensaje de los citados en la propiedad el antecedente de la misma sea falso antes

de la ejecución que la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. El cambio de sim_id_i que se produce al ejecutar $\pi_z = Abort_i$ va precedido de un incremento del valor de ta_i. Este efecto previo sobre el tiempo de activación del nodo i implica que el antecedente de la propiedad pasa a ser falso cuando en los mensajes considerados en la misma $first(m.path).id$ es i.

Por otra parte, la variable ta queda modificada al ejecutar $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ o $\pi_z = Abort_i$. En ambos casos, el tiempo de activación asociado al nodo i se incrementa de forma que $s_{z-1}.ta_i < s_z.ta_i$. Si el nodo correspondiente al primer elemento de las rutas de los mensajes considerados en la propiedad coincide con el nodo i, la propiedad se verifica porque el antecedente se hace falso. Si el nodo i aparece en cualquier otra posición de la ruta o no está incluido, el cambio de ta_i en s_z no afecta y la propiedad se cumple por inducción.

■

De forma parecida a la propiedad anterior, se garantiza que el campo *sid* de un mensaje *AVSRSP* o de un mensaje almacenado en *st_avsrsp* contiene la identidad simulada del último nodo de la ruta de esos mensajes. En este caso, el campo *sid* proporciona información, al primer nodo de la ruta de los mensajes, sobre la identidad simulada del candidato mayor que le sucede en la ruta considerada.

Propiedad B.2.23. Si $\exists\ \{i,\ j,\ x\} \in \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $path \in P^+$ verificando $(AVSRSP,\ t,\ sid,\ path{\cdot}(x,\ s.ta_x)) \in s.channel(j,\ i) \vee (t,\ sid,\ path{\cdot}(x,\ s.ta_x)) \in s.st\_avsrsp_i \Rightarrow sid = s.sim\_id_x$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \in \mathcal{N}$ se cumple que $s_0.channel(j,\ i) = \epsilon$ y $s_0.st\_avsrsp_i = NULL$ por lo que la propiedad es cierta.

La creación de un mensaje *AVSRSP*, m, es un posible efecto de las acciones: $\pi_z \in \{rcvINF_i(j,\ m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{INF}\}$, $\pi_z \in \{rcvAVS_i(j,\ m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ y $\pi_z = sndAVSRSP_i$. Las dos primeras acciones forman el nuevo mensaje a partir de un mensaje almacenado en $s_{z-1}.set\_st\_avs_i$ o un mensaje *AVS*, $m\prime$ que se retira del canal de comunicaciones respectivamente. En cambio, la acción del tipo $sndAVSRSP_i$ genera el mensaje combinando el contenido de $s_{z-1}.set\_st\_avs_i$ y $s_{z-1}.st\_avsrsp_i$.

■

Por último, esta propiedad indica que el identificador y el tiempo del campo *sid* de un mensaje *ALG* o un mensaje almacenado en *st_alg* coincide con el último para de la ruta que contienen. Esto implica que, la identidad del candidato que generó un mensaje *ALG* (último nodo de la ruta), se va transmitiendo y registrando entre sus predecesores directos e indirectos.

Propiedad B.2.24. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $path \in P^+$ verificando que $(ALG,\ t,\ sid,\ path) \in s.channel(j,\ i) \vee (sid,\ path) \in s.st\_alg_i \Rightarrow (sid.id,\ sid.t) = last(path)$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \in \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ y $s_0.channel(j,\ i) = \emptyset$ por lo que la propiedad es cierta.

Al ejecutar cualquiera de las siguientes acciones: $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_i$ y $\pi_z \in \{rcvINF_i(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$ y generarse un mensaje ALG, m, se puede asegurar que verifica la propiedad. El campo $m.sid$ es en todos los casos $s_z.sim\_id_i$ y el nodo i coincide con $last(m.path).id$ porque la ruta de los mensajes está formada sólo por el par $(i,\ s_z.ta_i)$. En el caso de la acción $\pi_z \in \{EndAddArc_i(j,\ t)\colon t \in \mathbb{N}\}$, si $s_{z-1}.status.alg_i = candidate$, el mensaje ALG cumple la propiedad al igual que en las acciones anteriores porque su ruta contiene únicamente el par $(i,\ s_z.ta_i)$. Sin embargo, cuando $s_{z-1}.status.alg_i = dummy$, el mensaje que se pone en el canal se crea a partir del contenido de $s_{z-1}.st\_alg_i$, de manera que $m.sid = s_{z-1}.st\_alg_i.sid$ y $last(m.path) = last(s_{z-1}.st\_alg_i.path)$. Así que, como la variable sim_id_i no se ve modificada por los efectos de la acción, la hipótesis inductiva permite demostrar que la propiedad se cumple en s_z para ese nuevo mensaje ALG.

Asimismo, existe otra acción generadora de mensajes ALG y que permite almacenar mensajes en st_alg_i. La ejecución de $\pi_z \in \{rcvALG_i(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ comprende distintos efectos según el estado del nodo i. Si $s_{z-1}.status.alg_i = blocked$, el mensaje ALG que se retira del canal se almacena en $s_z.st\_alg_i$ de manera que el campo sid toma el valor $m.sid$ y el elemento $(i,\ s_z.ta_i)$ antecede a $m.path$ en la nueva ruta. Además, en el caso de que el nodo i cumpla la condición para enviar un nuevo mensaje ALG, su componente sid y su ruta serán idénticas a las almacenadas previamente en $s_z.st\_alg_i$. En ambas situaciones, la hipótesis inductiva permite concluir que la propiedad es cierta en s_z porque el valor de la variable sim_id asociada al último elemento de la ruta del mensaje ALG inicial se conserva en todas las versiones de mensajes resultantes $(m.sid = s_{z-1}.sim\_id_{last(m.path).id}$, $last(m.path) = last(s_z.st\_alg_i.path)$ y $m.sid = s_z.st\_alg_i.sid)$. Suponiendo que $s_{z-1}.status.alg_i = candidate$ y se den las condiciones para que se almacene tal cual el mensaje ALG retirado del canal en $s_z.st\_alg_i$, la hipótesis inductiva vuelve a ser concluyente.

Considerando los efectos de las acciones que pueden cambiar la variable st_alg_i, se observa que, al ejecutar la acción $\pi_z \in \{StartDelArc_i(x,\ t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, se procede a cancelar el contenido de st_alg_i a la vez que sim_id adopta un nuevo valor. De igual manera, al ejecutarse $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, se modifica sim_id_i y se anula $s_z.st\_alg_i$. Estos efectos pueden hacer el antecedente de la propiedad falso. Otras acciones que eliminan el contenido de la variable st_alg_i, cuando el nodo i pasa a ser $victim$, son $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$. El antecedente en estos casos también podría llegar a ser falso.

Entre los efectos de la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ está el que hace $s_z.st\_alg_i = NULL$. Este valor, obviamente, podría hacer que el antecedente de la propiedad fuera falso. Otras posibles ejecuciones de esta acción no afectarían al cumplimiento de la propiedad. Como la estructura de los mensajes considerados y la variable *sim_id* no se modifica, la hipótesis inductiva es suficiente para validar la propiedad.

Por último, hay que mencionar que la acción $\pi_z = Abort_i$ introduce un nuevo valor en sim_id_i. La condición de habilitación permite descartar que haya algo almacenado en $s_{z-1}.st\_alg_i$ (propiedad B.2.12) y si hubiere algún mensaje ALG en el canal cumpliendo la propiedad en s_{z-1}, la hipótesis inductiva bastaría para asegurar que ese mensaje sigue verificando la propiedad.

■

En esta propiedad se confirma que el campo *sid* de un mensaje *AVSRSP* o un mensaje almacenado en *st_avsrsp* contiene una identidad simulada superior a la del nodo destino del mensaje o al nodo al que se asocia el almacén. Esta misma propiedad permite indicar que la identidad simulada registrada en el campo *sid* se corresponde a un candidato sucesor que es mayor que la identidad simulada del primer nodo de la ruta de los mensajes considerados.

Propiedad B.2.25. Si $\exists\,\{i, j\} \in \mathcal{N}$, $t \in \mathbb{N}$, $sid \in T$, $path \in P^+$ verificando: ($AVSRSP$, t, sid, $(i,\ s.ta_i)\cdot path) \in s.channel(j,i) \vee (t,\ sid,\ (i,\ s.ta_i)\cdot path) \in s.st\_avsrsp_i \Rightarrow sid > s.sim\_id_i$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \in \mathcal{N}$ se cumple que $s_0.st\_avsrsp_i = NULL$ y $s_0.channel(j,i) = \epsilon$, por lo que la propiedad es cierta.

En primer lugar, se va a analizar cómo influye en el cumplimiento de la propiedad la creación de mensajes *AVSRSP*. La acción $\pi_z = \{sndAVSRSP_j\}$ genera un mensaje *AVSRSP* cuya ruta se forma combinando adecuadamente las rutas correspondiente a mensajes almacenados en $s_{z-1}.set\_st\_avs_j$ y $s_{z-1}.st\_avsrsp_j$. De la estructura del nuevo mensaje se desprende que $first(m.path)$ coincide con el primer elemento de la ruta, *path*, que aporta el mensaje de $s_{z-1}.set\_st\_avs_i$ y que en $m.sid$ se copia $s_{z-1}.st\_avsrsp_i.sid$. Aplicando la propiedad B.2.22 al primer elemento de*path*, se obtiene que su campo *sid* es $s_z.sim\_id_{first(m.path).id}$, siempre y cuando $s_z.ta_{first(path).id}$ = *first(path).ta*. Por otro lado, según la propiedad B.2.20, $s_{z-1}.cand\_succ_i$ equivale a $s_{z-1}.st\_avsrsp_i.sid$. Sustituyendo estas relaciones en la condición de formación del mensaje *AVSRSP*, $s_{z-1}.cand\_succ_i > sid$, resulta que $m.sid > s_z.sim\_id_{first(m.path).id}$ o $m.sid > s_z.sim\_id_i$.

Por otra parte, las acciones $\pi_z \in \{rcvINF_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$ y $\pi_z \in \{rcvAVS_j(x,\ m\prime)\colon j \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ tienen un efecto similar consistente en

la generación de un mensaje *AVSRSP*, *m*. La ruta del mensaje en el primer tipo de acción coincide con la ruta de un mensaje almacenado en $s_{z-1}.set\_st\_avs_i$, *n*, siendo $first(m.path) = first(n.path)$ y $m.sid = s_z.sim\_id_j$. La condición para crearse este mensaje es que $s_z.sim\_id_j > n.sid$. Sólo si $s_z.ta_{first(n.path).id} = first(n.path).ta$, la propiedad B.2.22 puede asegurar que $n.sid = s_{z-1}.sim\_id_{first(n.path).id}$. Sustituyendo las expresiones anteriores, se obtiene la siguiente relación, $m.sid > s_z.sim\_id_{first(m.path).id}$. Tal y como se quería demostrar. En cambio, la ruta del mensaje del segundo tipo de acción coincide con la ruta del mensaje *AVS* recibido, *m′*, siendo $first(m.path).id = first(m'.path).id$. Además, $m.sid = s_{z-1}.sim\_id_j$ y la ejecución de la acción no modifica sim_id_j. La condición de formación del mensaje *AVSRSP* consiste en que $s_{z-1}.sim\_id_j > m'.sid$. Sólo si $s_z.ta_{first(m'.path).id} = first(m'.path).ta$, la propiedad B.2.22 asegura que $m'.sid = s_{z-1}.sim\_id_{first(m'.path).id}$. Sustituyendo adecuadamente las expresiones anteriores, se obtiene la siguiente relación, $m.sid > s_z.sim\_id_{first(m.path).id}$. De esta forma, la propiedad queda demostrada.

Seguidamente, se va a comprobar que la propiedad se cumple cuando la variable que se modifica es st_avsrsp_i. Un posible efecto del grupo de acciones $\pi_z \in \{rcvAVSRSP_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$ y $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ consiste en almacenar un mensaje en $s_z.st\_avsrsp_i$. Si el mensaje que se incluye procede de un mensaje *AVSRSP* (primera acción), los campos del mismo, incluida la ruta, no se modifican. Como el resto de variables de la propiedad conservan sus valores, la hipótesis inductiva permite llegar a la conclusión de que la propiedad se sigue cumpliendo. Cuando el mensaje que se guarda se crea a partir de un mensaje *ALG* (segunda acción), se antepone el par $(i,s_z.ta_i)$ a la ruta del mensaje *m.path*. Aplicando la propiedad B.2.24 que establece que $m.sid = s_{z-1}.sim\_id_{last(m.path).id}$ e incorporándola a la condición de almacenamiento del mensaje *ALG*, $m.sid > s_{z-1}.sim\_id_i$, se obtiene que $s_z.sim\_id_{last(m.path).id} > s_z.sim\_id_i$. De esta forma se demuestra que el consecuente de la propiedad es cierto en s_z.

Otra variable de la propiedad que puede cambiar su valor es ta_i. Cuando se ejecutan las acciones $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ y $\pi_z = Abort_i$, $s_{z-1}.ta_i < s_z.ta_i$. Esto quiere decir que, siempre que se ejecute alguna de estas acciones y el nodo *i* sea el identificador del primer elemento de la ruta de un mensaje *AVSRSP* presente en s_{z-1}, el antecedente de la propiedad se hará falso porque el tiempo del primer par de la ruta no coincidirá ya con $s_z.ta_i$.

En lo que respecta a la variable sim_id_i, se van a estudiar los efectos de las acciones $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ y $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. En el primer caso, el cambio en sim_id_i se produce cuando $s_{z-1}.status.alg_i = active$. En la segunda acción, la condición de habilitación indica que $s_{z-1}.status.id_i$ debe ser *unknown* o, lo que es lo mismo, según la propiedad B.2.15, $s_{z-1}.status.alg_i \in \{active, blocked\}$. La propiedad B.2.14 asegura que, en cualquiera de los estados posibles para el nodo *i* en s_{z-1}, st_avsrsp_i está vacío.

Suponiendo que exista un mensaje *AVSRSP* dirigiéndose al nodo i en s_{z-1}, se sabe que el primer nodo de su ruta es precisamente el nodo i (propiedad B.2.18). Cuando el nodo i se incorporó en la ruta, su estado pasó a ser *candidate* o *dummy*. Así que, antes de ejecutarse la acción, el nodo i debe activarse mediante la acción $EndDelArc_i(j)$. Al activarse, ta_i se incrementa, haciendo que el tiempo asociado al nodo i registrado en la ruta del mensaje no coincida con su tiempo de activación. Como los efectos de las acciones no retiran el mensaje *AVSRSP* del canal, se concluye que, al producirse un cambio en sim_id_i, la propiedad se cumple con antecedente y consecuente falsos. ■

La siguiente propiedad hace referencia a la variable temporal *t_unk*. Un nodo que se activa y no ha participado en la transmisión de la identidad simulada de otro candidato reinicializa su valor de *t_unk* y de *st_alg*. Por otro lado, si el nodo adquiere una nueva identidad simulada también adquiere los valores iniciales de esas dos variables.

Propiedad B.2.26. $\forall\ i \in \mathcal{N}$ se verifica que $s.t\_unk_i = -1 \Leftrightarrow s.st\_alg_i = NULL$

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.t\_unk_i = -1$ y $s_0.st\_alg_i = NULL$ por lo que la propiedad es cierta. Para demostrar la propiedad se van a comprobar los efectos de las acciones que modifican las variables t_unk_i y st_alg_i.

Al ejecutarse en algunos casos las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ y $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ y las acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ cuando $s_z.status.alg_i = victim$, se producen al mismo tiempo los siguientes efectos: $s_z.t\_unk_i = -1$ y $s_z.st\_alg_i = NULL$. Estos valores implican que la propiedad es cierta.

El grupo de acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ puede ejecutarse en otras circunstancias que hacen que $s_z.t\_unk_i$ adquiera el valor $s_z.ta_i$ y se almacene el mensaje m en $s_z.st\_alg_i$. Obviamente en estas situaciones la propiedad también se verifica.

■

La variable temporal *t_unk* de un nodo nunca puede contener un valor superior al del tiempo de activación del nodo, *ta*. Cada vez que un nodo se bloquea incrementa su tiempo de activación, pero en el proceso de activación el valor de *t_unk* se reinicializa al valor -1 impidiendo que supere al correspondiente *ta*.

Propiedad B.2.27. $\forall\ i \in \mathcal{N}$ se verifica que $s.t\_unk_i \leq s.ta_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.t\_unk_i = -1$ y $s_0.ta_i = 1$ por lo que la propiedad es cierta.

Considerando las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon\ j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{INF}\}$ que tienen como efecto que $s_z.t\_unk_i$ pase a valer -1, se puede asegurar que la variable ta_i no cambia al ejecutar las acciones mencionadas. Además, el tiempo de activación de todos los nodos del sistema inicialmente es 1 y las acciones del algoritmo sólo lo pueden incrementar.

Por otro lado, la variable ta_i ve incrementado su valor en una unidad cuando se ejecutan las acciones $\pi_z \in \{EndDelArc_i(j)\colon\ j \in \mathcal{N}\}$ o $\pi_z = Abort_i$. En consecuencia, $s_{z-1}.ta_i < s_z.ta_i$. Como los efectos de estas acciones no modifican t_unk_i, aplicando la propiedad inductiva se concluye.

Finalmente, cabe mencionar un efecto de la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{ALG}\}$ que iguala el valor de t_unk_i con el de ta_i. En esta situación es evidente que la propiedad se cumple. ■

En esta propiedad se establece que, para todos los nodos activos o que están bloqueados conociendo su identidad simulada, que el valor *t_unk* no coincide con su tiempo de activación.

Propiedad B.2.28. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = active \vee (s.status.alg_i = blocked \wedge s.status.id_i = known) \Rightarrow s.t\_unk_i \neq s.ta_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$, $s_0.t\_unk_i = -1$ y $s_0.ta_i = 1$, por lo que la propiedad es cierta.

Los efectos de las acciones: $\pi_z = Abort_i$, $\pi_z = initiate_i$, $\pi_z \in \{rcvALG_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{AVS}\}$ modifican la variable $status.alg_i$ de modo que $s_z.status.alg_i \neq \{active,\ blocked\}$, haciendo falso el antecedente.

La acción $\pi_z \in \{StartAddArc_i(j)\colon\ j \in \mathcal{N}\}$ también cambia el estado del nodo i. Cuando $s_z.status.alg_i$ se convierte en *candidate* o $s_z.status.alg_i = blocked$ siendo en ambos casos $s_{z-1}.status.id_i = unknown$, se razona igual que en las acciones comentadas anteriormente. En cambio, si al ejecutar la acción $s_{z-1}.status.id_i = known$ y $s_z.status.alg_i$ se transforma en *blocked*, el antecedente pasaría a ser cierto. Como la condición de habilitación de esta acción indica que $s_{z-1}.state_i = active$, o lo que es lo mismo según la propiedad B.3.3, $s_{z-1}.status.alg_i = active$. En tal caso la hipótesis inductiva indica que $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$) y las variables t_unk_i y ta_i no cambian tras ejecutarse la acción. Por consiguiente, en la situación que se analiza el consecuente sigue siendo cierto.

Análogamente, los efectos de la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon\ j \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden hacer que el nodo i alcance el siguiente estado: $s_z.status.alg_i = blocked$ y

$s_z.status.id_i = known$ (antecedente cierto). En este caso, el consecuente también es cierto porque $s_z.t\_unk_i = -1$. Por otra parte, la acción $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ no modifica la variable $status.alg_i$, pero un posible efecto conlleva que $s_z.t\_unk_i$ pase a valer -1. Este cambio evidencia que el consecuente es cierto porque ta_i es siempre mayor o igual que 1.

Si se analizan los efectos que modifican la variable ta_i, hay que considerar las acciones $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\$$ y $\pi_z = Abort_i$. Al ejecutarse ambas acciones $s_{z-1}.ta_i < s_z.ta_i$. La ejecución de la primera acción conlleva siempre un cambio de estado $s_z.status.alg_i = active$ (antecedente cierto). Al igual que antes, si $s_z.t\_unk_i$ pasa a valer -1, el consecuente de la propiedad será también cierto y la propiedad se cumplirá. Si, por el contrario, los efectos no modifican t_unk_i, la propiedad B.2.27 en s_{z-1}, $s_{z-1}.ta_i \geq s_{z-1}.t\_unk_i$) asegura que al incrementar ta_i se verifica que $s_z.ta_i > s_z.t\_unk_i$.

■

Al contrario que en la anterior propiedad, en ésta se asegura que la coincidencia de *t_unk* y el tiempo de activación de un nodo se produce cuando los nodos han recibido un mensaje *ALG* (nodos *dummy* o *candidate* con información en su correspondiente *st_alg*).

Propiedad B.2.29. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = dummy \vee (s.status.alg_i = candidate \wedge s.st\_alg_i \neq NULL) \Rightarrow s.t\_unk_i = s.ta_i$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ por lo que la propiedad se cumple.

Tras la ejecución de las acciones: $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z = Abort_i$, $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$, la variable $status.alg_i$ se puede modificar de manera que $s_z.status.alg_i \neq \{dummy, candidate\}$. Por tanto, la propiedad se verifica en estos casos con el antecedente falso.

Por otra parte, al ejecutarse $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ o $\pi_z = initiate_i$, $s_z.status.alg_i$ puede llegar a convertirse en *candidate*. Para que la primera acción esté habilitada y se alcance el estado *candidate*, el nodo i debe ser *active* y además $s_{z-1}.status.id_i = known$. La habilitación de la segunda acción precisa que $s_{z-1}.status.alg_i = blocked$ y $s_{z-1}.status.id_i = known$. De acuerdo con estas condiciones, la propiedad B.2.13 establece que, en ambos casos, $s_{z-1}.st\_alg_i = NULL$. Como los efectos de las acciones no modifican esta variable, $s_z.st\_alg_i$ sigue estando vacío. En consecuencia, se puede asegurar que la propiedad se verifica con el antecedente falso. Del mismo modo, un posible efecto de la acción $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ hace que $s_z.status.alg_i = candidate$. Como el cambio de estado del nodo i va acompañado de otro efecto por el que $s_z.st\_alg_i = NULL$, la propiedad también queda demostrada.

Los efectos de la acción $\pi_z = \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ no cambian el estado del nodo i, pero pueden en algún caso hacer que $s_z.st\_alg_i = NULL$ y $s_z.t\_unk_i = -1$. Como estos cambios se producen cuando $s_{z-1}.status.alg_i = active$ y ya se ha comentado que esta variable no se modifica en la ejecución de la acción, se concluye que la propiedad es cierta porque en esta situación el antecedente es falso en s_z.

Cuando se ejecuta la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y los estados resultantes del nodo i son *dummy* o *candidate*, se observa que, al mismo tiempo, st_alg_i pasa a contener un mensaje y t_unk_i adopta el valor de ta_i. Por consiguiente, en estas circunstancias, el antecedente y el consecuente son ciertos, verificándose así la propiedad.

En lo que respecta a los cambios en la variable ta_i que introducen las acciones $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ o $\pi_z = Abort_i$, sólo cabe mencionar que el incremento del tiempo de activación del nodo i supone que el consecuente de la propiedad es falso porque t_unk_i no cambia o adopta el valor -1. En cualquiera de estos dos casos, la propiedad se cumple en s_z debido a que el antecedente se hace falso (el nodo i alcanza el estado *active* y *aborted* respectivamente), tal y como se indicó anteriormente. ■

La siguiente propiedad garantiza que sólo los nodos *dummy* tienen información almacenada en su correspondiente *st_alg* y, además, el primer elemento de la ruta que almacenan coincide con su identificador y tiempo de activación.

Propiedad B.2.30. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = dummy \Leftrightarrow s.st\_alg_i \neq NULL \wedge first(s.st\_alg_i.path) = (i,\ s.ta_i)$

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ y $s_0.st\_alg_i = NULL$ por lo que la propiedad es cierta.

Entre los efectos de las acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ cabe destacar el que hace que $s_z.status.alg_i = victim$. En esta situación, además de cambiar el estado del nodo i, se vacía st_alg_i. Así pues, ambos lados de la implicación son falsos.

Tras la ejecución de la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ se elimina el contenido de st_alg_i y $s_z.status.alg_i \in \{active,\ blocked,\ candidate\}$. Esto implica que la propiedad se cumple.

La única variable de la propiedad que modifica la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ es $status.alg_i$. La condición de habilitación que marca el medio es $s_{z-1}.state_i = active$ o, usando la propiedad B.3.3, $s_{z-1}.status.alg_i = active$. La hipótesis inductiva indica ambos lados de la doble implicación son falsos. Aunque por los efectos de la acción resulta $s_z.status.alg_i \in \{blocked,\ candidate\}$, las variables st_alg_i y ta_i no se ven modificadas. De este modo, queda demostrada la propiedad.

Al ejecutar la acción $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ es posible que $s_z.st\_alg_i = NULL$. En ese caso, $s_{z-1}.status.alg_i = active$. Dado que $status.alg_i$ no

varía tras la ejecución de la acción, la propiedad se verifica.

Al ejecutarse la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$, $s_z.status.alg_i = active$. En los casos en los que $s_z.st\_alg_i = NULL$ es evidente que la propiedad se verifica porque ambos lados de la doble implicación son falsos. Sin embargo, en el resto de casos $s_z.st\_alg_i = NULL$, se usa el incremento del tiempo de activación del nodo i para confirmar que la propiedad se verifica. El cambio de ta_i hace que *first*$(s_z.st\_alg_i.path) \neq (i,\ s_z.ta_i)$.

La acción $\pi_z = Abort_i$ cambia $status.alg_i$ a *aborted*. De acuerdo con la propiedad B.2.12, en ese estado $s_z.st\_alg_i = NULL$.

La ejecución de la acción $\pi_z = initiate_i$ también modifica $status.alg_i$ y lo convierte en *candidate*. Para que la acción esté habilitada es preciso que $s_{z-1}.status.alg_i = blocked$ y $s_{z-1}.status.id_i = known$. Según la propiedad B.2.13, $s_{z-1}.st\_alg_i = NULL$. Como los efectos de las acciones no cambian esta variable, se concluye que la propiedad se cumple en s_z.

Cuando se ejecuta la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ es posible que $s_z.status.alg_i$ pase a ser *dummy*. Los efectos que acompañan este cambio de estado consisten en almacenar en $s_z.st\_alg_i$ una ruta que verifica que *first*$(s_z.st\_alg_i.path) = (i,\ s_z.ta_i)$. Por consiguiente, ambos lados de la doble implicación son ciertos. En el caso de que esta misma acción almacene información en st_alg_i siendo $s_z.status.alg_i = candidate$, hay que comprobar la estructura de la ruta que se almacena. Dado que la ruta almacenada es copia de la del mensaje ALG que se retira del canal y no se incorpora el nodo i que es el receptor del mensaje, se deduce que *first*$(s_z.st\_alg_i.path) \neq (i,\ s_z.ta_i)$. En este supuesto la propiedad también se cumple ya que ambos lados de la doble implicación son falsos.

■

En esta propiedad se confirma que un nodo que desconoce su identidad simulada en un proceso de borrado de esperas (*unknown*), mantiene el contenido de su almacén *st_alg* aunque su tiempo de activación deja de coincidir con el valor de la variable *t_unk* asociada a él porque se ha desbloqueado.

Propiedad B.2.31. $\forall\ i \in \mathcal{N}$ se verifica que $s.st\_alg_i \neq NULL \wedge s.t\_unk_i \neq s.ta_i \Leftrightarrow s.status.id_i = unknown$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ y $s_0.status.id_i = known$, por lo que la propiedad es cierta.

Si se ejecuta $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ resulta que $s_z.st\_alg_i = NULL$ y $s_z.status.id_i = known$ con lo que ambos lados de la implicación son falsos.

Por otra parte, el efecto de las acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ hacen que $s_z.status.alg_i = victim$ y que,

por la propiedad B.2.15, va seguido de otros efectos que modifican las variables del antecedente de la propiedad, esto es, $s_z.st\_alg_i = NULL$ y $s_z.t\_unk_i = -1$.

De manera similar, al ejecutarse la acción $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ se puede llegar a cancelar el contenido de st_alg_i. Para la primera acción este efecto tiene lugar siendo $s_{z-1}.status.id_i = known$. En las ejecuciones de la segunda acción, ese efecto se obtiene cuando $s_{z-1}.status.alg_i \in \{dummy,candidate\}$. La propiedad B.2.15 establece que en esos dos posibles estados también $s_{z-1}.status.id_i = known$. Como la variable $status.id_i$ no cambia en la ejecución de las acciones citadas, queda demostrada la propiedad.

El único efecto de la acción $\pi_z = Abort_i$ que modifica las variables que intervienen en la propiedad es el incremento de ta_i. Aunque el cambio en el tiempo de activación del nodo i hace que $s_z.t\_unk_i \neq s_z.ta_i$, al ejecutarse esta acción también cambia $status.alg_i$ pasando a ser *aborted*. Según la propiedad B.2.12 y la propiedad B.2.15, en ese estado se concluye que $s_z.st\_alg_i = NULL$ y $s_z.status.id_i = known$. En consecuencia, la propiedad se verifica.

Aparte de la ejecución mencionada de la acción $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$, existen otras posibilidades que provocan que la variable t_unk_i tome el valor actual del tiempo de activación del nodo i y simultáneamente se incluya información en st_alg_i. Como estos efectos tienen lugar en caso de que $s_{z-1}.status.id_i = known$ y esta variable no varía en la ejecución, se concluye que la propiedad se cumple en s_z.

Hay casos en los que la ejecución de la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ hace que $s_z.status.id_i = unknown$. Si el efecto se produce siendo $s_{z-1}.status.alg_i = dummy$, se sabe por la propiedad B.2.30 que $s_{z-1}.st\_alg_i \neq NULL$. Por otra parte, la propiedad B.2.29 señala que $s_{z-1}.t\_unk_i = \mathrm{s}_{z-1}.\mathrm{ta}_i$. Como el tiempo de activación del nodo i se incrementa resultando $s_z.t\_unk_i \neq s_z.ta_i$ y tanto st_alg_i como t_unk_i no varían en este supuesto, se concluye que la propiedad se cumple. Cuando $s_{z-1}.status.alg_i = candidate$ y $s_{z-1}.inf\_need_i = true$ tambiíen se realiza el cambio de $status.id_i$ a *unknown*. La propiedad B.2.35 asegura que $s_{z-1}.st\_alg_i \neq NULL$. De nuevo, según la propiedad B.2.29, se establece que $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$. Igual que en el caso anterior, el incremento de ta_i y la conservación de los valores de st_alg_i y t_unk_i, permiten concluir.

■

Con esta propiedad se aclara cómo se va formando de la ruta de un mensaje ALG conforme éste se va propagando entre los predecesores del nodo que lo generó inicialmente. El identificador del nodo que envía un mensaje ALG siempre aparece en el primer par de la ruta.

Propiedad B.2.32. $\forall\ \{i,\ j\} \subseteq \mathcal{N}$ se verifica que $\exists\ m \in M_{ALG}\colon m \in s.channel(i,\ j) \Rightarrow first(m.path).id = i$

Demostración: En el estado inicial, s_0, $\forall$ $\{i,j\} \in \mathcal{N}$ se cumple que $s_0.channel(i, j) = \epsilon$ por lo que la propiedad es cierta. A continuación se analizan los efectos de las acciones que pueden generar un mensaje ALG.

La ejecución de las acciones: $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_i(j, t)\colon t \in \mathbb{N}\}$ cuando $s_{z-1}.status.alg_i = candidate$, $\pi_z = initiate_i$, $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, lleva consigo la formación de un mensaje ALG, $m\prime$, que cumple que $first(m\prime.path).id = i$ tal y como se señala en la propiedad. Para comprobar la propiedad cuando se ejecuta la acción $\pi_z \in \{EndAddArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ siendo $s_{z-1}.status.alg_i = dummy$ es necesario confirmar que $first(s_{z-1}.st\_alg_i.path).id = i$. Como la ruta del nuevo mensaje ALG se construye a partir de la ruta contenida en $s_{z-1}.st\_alg_i$, la propiedad B.2.30 asegura que si $s_{z-1}.status.alg_i = dummy$ entonces $s_{z-1}.st\_alg_i \neq NULL$ y $first(s_{z-1}.st\_alg_i.path) = (i, s_{z-1}.ta_i)$. Por consiguiente, resulta evidente que $first(m\prime.path).id = i$ y la propiedad también se verifica.

Por otra parte, la sola retirada del canal de un mensaje ALG por efecto de la acción $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ podría llegar a hacer el antecedente de la propiedad falso.

■

Esta propiedad permite afirmar que los nodos que tienen almacenada información en su *st_avsrsp*, o su estado es *victim*, o conocen a un candidato sucesor y tienen también información almacenada en su correspondiente *st_alg*.

Propiedad B.2.33. $\forall$ $i \in \mathcal{N}$ se verifica que $s.st\_avsrsp_i \neq NULL \Rightarrow (s.st\_alg_i \neq NULL \wedge s.cand\_succ_i \neq NULL) \vee s.status.alg_i = victim$.

Demostración: En el estado inicial, s_0, $\forall$ $i \in \mathcal{N}$ se cumple que $s_0.st\_avsrsp_i = NULL$ por lo que la propiedad es cierta. Para demostrar que la propiedad se cumple en s_z se va a proceder a estudiar los efectos que modifican cualquiera de las variables que ésta incluye.

Tanto el grupo de acciones $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ como $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ permite que el estado del nodo i pase a ser *victim*. Al alcanzar este estado es obvio que la propiedad se cumple con el consecuente cierto.

La precondición de la acción $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ indica que $s_{z-1}.status.id_i = unknown$. Esto conlleva, según la propiedad B.2.15, que $s_{z-1}.status.alg_i \in \{active, blocked\}$. En cualquiera de esos estados, la propiedad B.2.14 señala que $s_{z-1}.st\_avsrsp_i = \emptyset$. Tras la ejecución de la acción, $s_z.st\_avsrsp_i$ no se modifica. En consecuencia, el antecedente es falso y la propiedad se cumple.

La acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ está habilitada si $s_{z-1}.state_i = active$, esto es, $s_{z-1}.status.alg_i = active$ (propiedad B.3.3).

Del mismo modo, la condición de habilitación de la acción $\pi_z = initiate_i$ requiere que $s_{z-1}.status.alg_i = blocked$. Considerando la propiedad B.2.14 en ese estado, se sabe que $s_{z-1}.st\_avsrsp_i = \emptyset$. Al ejecutarse la acción no se modifica esta variable y, por tanto, la propiedad se verifica con el antecedente falso. En el caso de que se ejecute la acción $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ siendo $s_{z-1}.status.alg_i = blocked$ o la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ siendo $s_{z-1}.status.alg_i = dummy$, se razona de igual manera.

La acción $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ no modifica ninguna de las variables de la propiedad a excepción de st_alg_i que puede vaciarse. En esta situación, $s_{z-1}.status.alg_i = active$ y, de acuerdo con la propiedad B.2.14, se asegura que $s_{z-1}.st\_avsrsp_i = NULL$. Por consiguiente, el antecedente es falso en s_z.

Por otra parte, tanto la acción $\pi_z = Abort_i$ como la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ cuando $s_{z-1}.status.alg_i = candidate$, tienen como efecto la eliminación del contenido de st_avsrsp_i. Esto implica que ambas cumplen la propiedad en s_z siendo el antecedente falso.

Las únicas acciones que añaden información a st_avsrsp_i son: $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z = \{rcvAVSRSP_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$. En la ejecución de la primera acción, además de resultar que $s_z.st\_avsrsp_i \neq NULL$, las variables del consecuente adquieren valores de forma que, $s_z.st\_alg_i \neq NULL$ y $s_z.cand\_succ_i \neq NULL$. En consecuencia, antecedente y consecuente pasan a ser ciertos. Cabe mencionar que estos últimos efectos se pueden producir sin incluir ningún contenido en st_avsrsp_i. En ese caso la propiedad también se cumple porque el consecuente se hace cierto en s_z.

Respecto a la segunda acción, al ejecutarse se incorpora información tanto en $s_z.cand\_succ_i$ como st_avsrsp_i. Para demostrar la propiedad es necesario confirmar que $s_z.st\_alg_i \neq NULL$. Dado que los efectos citados se producen cuando $s_{z-1}.status.alg_i = candidate$ y le llega un mensaje $AVSRSP$, se asume que la existencia de ese mensaje es debida a que intercambió información con el candidato sucesor que lo envía. Por tanto, conoció a su candidato sucesor a través de un mensaje ALG, que quedó almacenado en $s_{z-1}.st\_alg_i$.

■

Los nodos en estado *dummy*, *blocked*, *active*, *victim* que conocen su identidad simulada, no necesitan información de nodos *candidate* que le suceden cuando se ha producido un borrado de esperas que los desvinculan de esos *candidate*.

Propiedad B.2.34. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i \in \{dummy, blocked, active, victim\} \wedge s.status.id_i = known \Rightarrow s.inf\_need_i = false$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$, $s_0.status.id_i = known$ y $s_0.inf\_need_i = false$ por lo que la propiedad es cierta.

De todas las acciones del algoritmo, las únicas que al ejecutarse pueden modificar la variable inf_need_i son: $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$, $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, $\pi_z = sndAVS_i$ y $\pi_z = sndAVSRSP_i$. En las cuatro primeras acciones $s_z.inf\_need_i$ pasa a ser *false*, lo que implica que la propiedad se cumple con el consecuente cierto. Por el contrario, los efectos de las dos últimas acciones hacen que $s_z.inf\_need_i$ sea *true*. En estas circunstancias es preciso que el antecedente de la propiedad en s_z también sea falso para que la propiedad quede demostrada. La habilitación de estos dos tipos de acciones requiere que $s_{z-1}.cand\_succ_i \neq NULL$. De acuerdo con la propiedad B.2.17, esa condición conlleva que $s_{z-1}.status.alg_i$ sea *candidate*. Como la variable $statu.alg_i$ no es modificada por los efectos de estas acciones, $s_z.status.alg_i = candidate$ confirmando que el antecedente es falso.

En lo que concierne a la variable *status.id*, las acciones $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ y $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden cambiar su valor. Tras la ejecución de la primera acción puede resultar que $s_z.status.id_i$ sea *unknown* (antecedente falso). En cambio, cualquier posible ejecución de la segunda acción hace que $s_z.status.id_i$ sea *known*. La condición de habilitación supone que $s_{z-1}.status.id_i = unknown$ y, según la propiedad B.2.15, eso se corresponde con $s_{z-1}.status.alg_i \in \{blocked, active\}$. En los casos que esos estados se mantienen tras la ejecución, el antecedente de la propiedad pase a ser cierto al ejecutar la acción. El valor de $s_{z-1}.inf\_need_i$ puede ser *true* o *false*, pero por efecto de la acción el valor *true* siempre se convierte en *false*. En cualquier caso, el consecuente de la propiedad en s_z es cierto, por lo que la propiedad se verifica.

Al ejecutarse otro grupo de acciones se producen cambios en $status.alg_i$. Los efectos de la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ pueden transformar el estado del nodo i en *blocked*. Sólo si $s_z.status.id_i$ es *known* es necesario comprobar el consecuente. Para que se dé el efecto indicado $s_{z-1}.status.alg_i = active$ y $s_{z-1}.status.id_i = known$. Esto implica, aplicando la hipótesis inductiva, que $s_{z-1}.inf\_need_i = false$. Como la variable inf_need_i no se modifica al producirse el efecto indicado, el consecuente sigue siendo cierto y la propiedad se cumple.

Respecto a la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$, en su ejecución se convierte $s_z.status.alg_i$ a *active*. Si en el estado previo $s_{z-1}.status.alg_i = dummy$, la propiedad B.2.15 establece que $s_{z-1}.status.id_i = known$. Aplicando la propiedad inductiva, se sabe que $s_{z-1}.inf\_need_i$ entonces vale *false*. En caso de que, tras la ejecución de la acción, se conserve el valor de $status.id_i$, resulta que la propiedad es cierta porque inf_need_i tampoco se altera. Cuando esta misma acción se ejecuta siendo $s_{z-1}.status.alg_i = candidate$ y $s_{z-1}.inf\_need_i = false$, $s_z.status.alg_i$ pasa a *active* pero inf_need_i no cambia de valor, quedando demostrada la propiedad.

Por último, cuando se ejecuta la acción $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ siendo $s_{z-1}.status.alg_i = blocked$ y resulta que $s_z.status.alg_i$ pasa a ser *dummy*,

la hipótesis inductiva concluye porque el resto de variables de la propiedad no varían.

Si se analizan otros cambios posibles en $status.alg_i$, se observa que tanto la acción $\pi_z = Abort_i$ como la acción $\pi_z = initiate_i$ permiten alcanzar los estados *aborted* y *candidate* respectivamente. Estos valores evidencian que el antecedente de la propiedad en s_z es falso. Del mismo modo, los efectos de las acciones $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ y $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden provocar que $s_z.status.alg_i$ pase a *candidate*. La propiedad en estos casos también se cumple.

■

Todo nodo candidato que ha enviado mensajes *AVS* o *AVSRSP*, combinando la información de sus almacenes, conserva el contenido de *st_alg* ya que en él que han quedado registrados los pares de la ruta por los que se difundió un mensaje *ALG*.

Propiedad B.2.35. $\forall\ i \in \mathcal{N}$ se verifica que $s.inf\_need_i = true \Rightarrow s.st\_alg_i \neq NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.inf\_need_i = false$ por lo que la propiedad es cierta.

Las acciones $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ cuentan entre sus efectos uno que modifica el valor de la variable inf_need_i. Cuando se ejecutan todas estas acciones $s_z.inf\_need_i$ adquiere el valor *false*. Como al mismo tiempo se produce otro efecto que cancela el contenido de st_alg_i en s_z, la propiedad se cumple con el antecedente y el consecuente falsos.

Por otro lado, tras la ejecución de las acciones $\pi_z = sndAVS_i$ o $\pi_z = sndAVSRSP_i$, $s_z.inf\_need_i$ pasa a ser *true*. La precondición de ambas acciones indica que $s_{z-1}.st\_avsrsp_i \neq NULL$. De acuerdo con la propiedad B.2.33, si $s_{z-1}.st\_avsrsp_i \neq NULL$ entonces también $s_{z-1}.st\_alg_i \neq NULL$. Como ninguna de las acciones modifica la variable st_alg_i, el consecuente es cierto y, por tanto, la propiedad se cumple.

En lo que respecta a efectos que introducen información en st_alg_i, sólo la acción $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ lo permite. Obviamente cuando se ejecuta esta acción en estos supuestos, la propiedad se verifica con el consecuente cierto.

Finalmente, la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ puede vaciar el contenido de st_alg_i sin que la variable inf_need_i varíe. Esta situación puede darse o bien cuando $s_{z-1}.status.alg_i$ sea *candidate* y $s_{z-1}.inf\_need_i$ *false* o bien cuando $s_{z-1}.status.alg_i$ sea *dummy* y $s_{z-1}.setPredToInf_i \neq \emptyset$. En el primer caso es obvio que la propiedad se cumple en s_z porque el antecedente es falso. Considerando el otro caso posible, basta con asegurar que si el nodo i en el estado previo es *dummy*, entonces $s_{z-1}.inf\_need_i = false$. La propiedad B.2.34 confirma este valor de la variable.

■

La siguiente propiedad indica que los nodos candidatos que han enviado mensajes *AVS* o *AVSRSP*, combinando la información de sus almacenes, mantienen el contenido correspondiente a su *st_avsrsp*.

Propiedad B.2.36. $\forall\ i \in \mathcal{N}$ se verifica que $s.inf\_need_i = true \wedge s.status.id_i = known \Rightarrow s.st\_avsrsp_i \neq NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.inf\_need_i = false$ por lo que la propiedad es cierta.

Entre los posibles efectos de las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ existe uno por el que la variable $s_z.inf\_need_i$ adquiere el valor *false*. En consecuencia, la propiedad en s_z se verifica con el antecedente falso.

Por el contrario, las acciones $\pi_z = sndAVS_i$ y $\pi_z = sndAVSRSP_i$ cambian el valor de inf_need_i a *true*. Para que estas acciones estén habilitadas es preciso que $s_{z-1}.st\_avsrsp_i \neq NULL$. Como el resto de efectos de la acción no cambia el contenido de st_avsrsp_i, se confirma que la propiedad se cumple en s_z porque el consecuente es cierto.

Considerando la variable *st_avsrsp*, los efectos de las acciones $\pi_z \in \{rcvALG_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVSRSP_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$ son los únicos que pueden incluir información en ella, $s_z.st\_avsrsp_i \neq NULL$. Así que, al ser el consecuente cierto en estos casos, la propiedad se verifica en s_z.

Contrariamente, tras la ejecución de la acción $\pi_z = Abort_i$ resulta que $s_z.st\_avsrsp_i = NULL$ (consecuente falso). La precondición de la acción señala que $s_{z-1}.status.alg_i$ debe ser *victim*. Según la propiedad B.2.34, en ese estado $s_{z-1}.inf\_need_i$ tiene que valer *false*. Dado que los efectos de la acción no modifican la variable inf_need_i, se concluye que la propiedad se cumple en s_z con el antecedente falso.

Para finalizar se va a estudiar la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ que junto con la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ son las que pueden cambiar la variable $status.id_i$. Al ejecutarse la primera acción se puede conseguir que $s_z.status.id_i$ pase a ser *unknown* (antecedente falso). También es posible que, siendo $s_{z-1}.status.alg_i = candidate$, se vacíe el contenido de $s_z.st\_avsrsp_i$. Si $s_{z-1}.inf\_need_i = true$, el cumplimiento de la propiedad es obvio porque la ejecución lleva consigo otro efecto tal que $s_z.status.id_i = unknown$. En caso de que $s_{z-1}.inf\_need_i$ sea *false* el antecedente de la propiedad será falso debido a que la ejecución de la acción no modifica esta variable. En lo que se refiere a la acción $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$, $status.id_i$ se convierte en *known* cuando se ejecuta. Como $s_z.inf\_need_i$ también pasa a valer *false* en cualquier situación, queda probado que el antecedente es falso. ■

Esta propiedad confirma que todo nodo cuyo *t_unk* sea -1 y no tenga información almacenada en su correspondiente *st_alg* es un nodo que se ha activado. En esa situación tiene conocimiento pleno de su identidad simulada y está en condiciones de participar en una nueva detección si se vuelve a bloquear.

Propiedad B.2.37. $\forall\ i \in \mathcal{N}$ se verifica que $s.t\_unk_i = -1 \wedge s.st\_alg_i = NULL \Rightarrow s.status.id_i = known$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.t\_unk_i = -1$, $s_0.st\_alg_i = NULL$ y $s_0.status.id_i = known$.

La única acción que puede cambiar a *unknown* el valor de la variable $status.id_i$ es $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$. Este efecto se consigue en dos supuestos distintos, esto es, si $s_{z-1}.status.alg_i = dummy$ o cuando $s_{z-1}.status.alg_i = candidate$ y $s_{z-1}.inf\_need_i = true$. En la primera situación la propiedad B.2.30 y en la segunda la propiedad B.2.35 establecen que $s_{z-1}.st\_alg_i \neq NULL$. Al producirse el efecto comentado sobre $status.id_i$, la variable st_alg_i no cambia. En consecuencia, tanto el antecedente como el consecuente son falsos y la propiedad queda probada.

Cuando $s_z.status.alg_i$ se convierte en *victim* al ejecutar las acciones $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVS_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{AVS}\}$, $s_z.t\_unk_i = -1$ y $s_z.st\_alg_i = NULL$. Ambas acciones precisan que $s_{z-1}.status.id_i$ sea *known* para estar habilitadas. Como la variable $status.id_i$ no se modifica en la ejecución de estas acciones, se concluye que la propiedad se cumple en s_z. Del mismo modo, al ejecutarse la acción $\pi_z \in \{EndDelArc_i(j)\colon j \in \mathcal{N}\}$ o $\pi_z \in \{StartDelArc_i(j, t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ el antecedente de la propiedad se hace cierto sin modificar el valor de $status.id_i$ que en este caso también debe ser *known* en s_{z-1}.

Por último, cabe mencionar que al ejecutarse la acción $\pi_z \in \{rcvINF_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ el antecedente y el consecuente pasan a ser ciertos en s_z. ■

En esta propiedad se relaciona el envío de un mensaje *ALG* con las variables que describen la espera que une al nodo emisor y receptor del mismo. Se concluye que la espera puede estar totalmente formada, parcialmente borrada o parcialmente borrada debido al aborto del emisor del mensaje que detectó y resolvió un interbloqueo.

Propiedad B.2.38. Sea $\{i, j\} \subseteq \mathcal{N}$, se verifica que si $\exists\ m \in M_{ALG}\colon m \in s.channel(j, i) \wedge m.ta = s.ta_i \Rightarrow (i,\ s.ta_i,\ received) \in s.set\_waiters_j \vee (i,\ s.ta_i,\ released) \in s.set\_waiters_j \vee s.status.alg_j = aborted$.

Demostración: En el estado inicial, s_0, $\forall\ \{i, j\} \subseteq \mathcal{N}$ se cumple que $s_0.channel(j, i) = \epsilon$.

- $\pi_z \in \{EndAddArc_j(i, t)\colon t \in \mathbb{N}\}$. Observando los efectos que generan un mensaje ALG, se puede asegurar que en esos casos $(i, t) \in s_z.setPred_j$, o lo que es lo mismo por la propiedad B.3.2, $(i, t) \in s_z.setPred_j$. Si $t = s_z.ta_i$ la propiedad se verifica con el consecuente cierto. En cambio, cuando $t \neq s_z.ta_i$, la propiedad se cumple con el antecedente falso. En el caso de que los efectos de la acción sólo incorporen $(i, t) \in s_z.setPred_j$, la propiedad es cierta por hipótesis inductiva.

- $\pi_z \in \{StartDelArc_j(i, t)\colon t \in \mathbb{N}\}$. Para que la acción esté habilitada $(i, t, received) \in s_{z-1}.set\_waiters_j$. Si se ejecuta la acción siendo $s_{z-1}.state_j = active$, la tupla de $set\_waiters_j$ se sustituye por $(i, t, released)$. En caso de que $t = s_{z-1}.ta_i$ el consecuente de la propiedad es cierto en s_z. Si $t \neq s_{z-1}.ta_i$, se concluye por hipótesis inductiva.

  Cuando $s_{z-1}.state_i = aborted$, la propiedad B.1.4 señala que $s_{z-1}.t\_activ_i > t$ o, aplicando la propiedad B.3.1, $t < s_{z-1}.ta_i$. Dado que los efectos de la acción no cambian la variable ta_i, la hipótesis inductiva permite concluir en este supuesto.

- $\pi_z = EndDelArc_i(j)$ o $\pi_z = Abort_i$. Uno de los efectos de estas acciones consiste en el incremento del tiempo de activación del nodo i de manera que $s_{z-1}.ta_i < s_z.ta_i$. Si en s_{z-1} existe un mensaje ALG que se dirige al nodo i se sabe por la propiedad B.2.2 que $m.ta \leq s_{z-1}.ta_i$. Cuando se produce el efecto mencionado, el antecedente de la propiedad resulta ser falso en cualquier situación porque $m.ta < s_z.ta_i$.

- $\pi_z = Abort_j$. Al ejecutarse la acción $s_z.status.alg_j = aborted$. El resto de acciones que modifican $status.alg_j$ no es necesario analizarlas porque ninguna permite cambiar el estado del nodo j de *aborted* a otro estado distinto.

- $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$ o $\pi_z = initiate_j$ o $\pi_z \in \{rcvALG_j(x,m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ o $\pi_z \in \{rcvINF_j(x,m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$. Entre los posibles efectos de estas acciones se encuentra la formación del mensaje ALG, m, al que hace referencia el enunciado de la propiedad. El mensaje creado tiene como destino el nodo i y $m.ta = t$, siendo el par (i, t) un elemento de $s_{z-1}.setPred_j$. Según la propiedad B.3.2, esto equivale a que la tupla $(i, t, received)$ pertenece a $s_{z-1}.set\_waiters_j$. Como los efectos de todas estas acciones no afectan al contenido del conjunto $set\_waiters_j$ ni a la variable ta_i, la propiedad se cumplirá con el consecuente cierto siempre que t coincida con $s_{z-1}.ta_i$. En el caso de que t sea distinto del valor $s_{z-1}.ta_{id}$, el antecedente de la propiedad será falso. Por otra parte, hay que mencionar que la acción $\{rcvALG_j(x, m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ se encarga de retirar del canal los mensajes de tipo ALG. Considerando este efecto separadamente, resulta obvio que la propiedad se verifica.

■

De forma similar, se describe el estado de una relación de espera entre dos nodos cuando uno de ellos ha recibido un mensaje ALG y lo tiene almacenado. Además si el nodo receptor de ese mensaje sigue bloqueado por el emisor del mismo, se sabe que la espera está completa, parcialmente borrada o el nodo emisor ha abortado porque formaba parte de un ciclo de esperas en el que no está el receptor.

Propiedad B.2.39. Sea $\{i, j\} \subseteq \mathcal{N}$, se verifica que si $s.st\_alg_i \neq NULL \wedge s.t\_unk_i = s.ta_i \wedge s.blocker_i = j \Rightarrow (i, s.ta_i, received) \in s.set\_waiters_j \vee (i, s.ta_i, released) \in s.set\_waiters_j \vee s.status.alg_j = aborted$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ lo que implica que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_i(x)\colon\ x \in \mathcal{N}\}$. La condición de habilitación de la acción indica que $s_{z-1}.state_i = active$ o, según la propiedad B.3.3, $s_{z-1}.status.alg_i = active$. En el caso de que $s_{z-1}.status.id_i = known$, la propiedad B.2.13 señala que $s_{z-1}.st\_alg_i = NULL$. Si, por el contrario, $s_{z-1}.status.id_i = unknown$, al aplicar la propiedad B.2.28, se sabe que $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$. Como los efectos de la acción mantienen el valor de todas esas variables, el antecedente de la propiedad es falso en s_z.

- $\pi_z \in \{EndAddArc_j(i,t)\colon\ t \in \mathbb{N}\}$. Cuando se ejecuta esta acción, $(i, t, received) \in s_z.set\_waiters_j$. Si t vale $s_z.ta_i$, el consecuente es cierto. En cualquier otro caso la hipótesis inductiva permite afirmar que la propiedad se cumple.

- $\pi_z \in \{StartDelArc_j(i,t)\colon\ t \in \mathbb{N}\}$. Si la acción está habilitada porque $s_{z-1}.state_j = active$ y $(i, t, received) \in s_{z-1}.set\_waiters_j$, el consecuente de la propiedad es cierto en s_{z-1} cuando $t = s_{z-1}.ta_i$. Al ejecutarse la acción se sustituye la terna $(i, t, received)$ por $(i, t, released)$. En consecuencia, el consecuente sigue siendo cierto porque la variable ta_i no varía. En cambio, si la acción se ejecuta siendo $s_{z-1}.state_i = aborted$, la propiedad B.1.1 establece que $s_{z-1}.blocker_i = NULL$. Al ejecutarse la acción, la variable $blocker_i$ sigue sin adquirir un valor. Por consiguiente, la propiedad es cierta con el antecedente falso.

- $\pi_z \in \{EndDelArc_i(j)\}$. Los efectos de la acción hacen que $blocker_i = NULL$.

- $\pi_z = Abort_i$. Para que esta acción se ejecute es necesario que $s_{z-1}.status.alg_i = victim$. De acuerdo con la propiedad B.2.12, en ese estado $s_{z-1}.st\_alg_i = NULL$. Como los efectos de la acción no modifican st_alg_i, la propiedad se verifica.

- $\pi_z = Abort_j$. Al ejecutarse la acción $s_z.status.alg_j = aborted$.

- $\pi_z = initiate_i$. La condición de habilitación de la acción señala que $s_{z-1}.status.alg_i = blocked$ y $s_{z-1}.status.id_i = known$. Aplicando la propiedad B.2.13, resulta que, en estas condiciones, $s_{z-1}.st\_alg_i = NULL$. Al ejecutar la acción la variable st_alg_i no varía. Por tanto, sea, la propiedad se cumple en s_z.

- $\pi_z \in \{rcvALG_i(j,m)\text{: } m \in M_{ALG}\}$. Algunos efectos de esta acción introducen información en st_alg_i y $s_z.t\_unk_i$ adopta el valor $s_z.ta_i$. El contenido de $s_z.st\_alg_i$ procede del mensaje ALG, m, que se retira del canal al ejecutar la acción. Dado que $m.ta$ coincide con $s_{z-1}.ta_i$, la propiedad B.2.38 establece que o bien la tupla $(i,\ s_{z-1}.ta_i,\ received)$ o $(i,\ s_{z-1}.ta_i,\ released)$ pertenecen a $s_{z-1}.set\_waiters_j$ o bien $s_{z-1}.status.alg_j = aborted$. Como la ejecución de la acción no cambia ni $set\_waiters_j$, ni $status.alg_j$, ni ta_i, se puede afirmar que el consecuente de la propiedad es cierto en s_z. Considerando otro posible efecto de esta acción que consiste en vaciar el contenido de $s_z.st\_alg_i$, se concluye que la propiedad también se cumple.

- $\pi_z \in \{rcvINF_i(x,m)\text{: } x \in \mathcal{N} \wedge m \in M_{INF}\}$. Tras la ejecución de la acción $s_z.st\_alg_i = NULL$.

- $\pi_z \in \{rcvAVS_i(x,m)\text{: } x \in \mathcal{N} \wedge m \in M_{AVS}\}$. Un posible efecto de esta acción consiste en que $s_z.st\_alg_i = NULL$, lo que implica que la propiedad se verifica.

- Entre los efectos de las acciones: $\pi_z \in \{StartAddArc_j(x)\text{: } x \in \mathcal{N}\}$, $\pi_z \in \{EndDel\ Arc_j(x)\text{: } x \in \mathcal{N}\}$, $\pi_z = initiate_j$, $\pi_z \in \{rcvALG_j(x,m)\text{: } x \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvINF_j(x,m)\text{: } x \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z \in \{rcvAVS_j(x,m)\text{: } x \in \mathcal{N} \wedge m \in M_{AVS}\}$, es posible el cambio de $status.alg_j$. Sin embargo, la hipótesis inductiva concluye porque $s_{z-1}.status.alg_j \neq aborted$ y los efectos no modifican esa situación.

■

La siguiente propiedad asegura que, cuando un mensaje ALG se dirige a un nodo con el tiempo correcto, ese nodo no puede formar parte del conjunto de predecesores de ningún otro nodo que no sea el emisor del mensaje.

Propiedad B.2.40. Sea $\{i,\ j\} \subseteq \mathcal{N}$, se verifica que si $\exists\ m \in M_{ALG}$ tal que $m \in s.channel(j,\ i) \wedge m.ta = s.ta_i \Rightarrow \forall\ k \neq j\text{: } (i,\ s.ta_i) \notin s.setPred_k$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \subseteq \mathcal{N}$ se cumple que $s_0.channel(j,\ i) = \epsilon$, por lo que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$ o $\pi_z = initiate_j$ o $\pi_z \in \{rcvALG_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z \in \{rcvINF_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Los efectos de estas acciones pueden generar un mensaje ALG, m, dirigido al nodo i con $m.ta = t$. Si el par $(i, t) \in s_{z-1}.setPred_i$ y $t = s_{z-1}.ta_i$, como no se modifica ta de nigún nodo al ejecutar la acción, se tiene que $m.ta = s_z.ta_i$ (antecedente cierto). Además, considerando la propiedad B.2.3, se cumple que $\forall\ k \neq x\ (i, t) \notin s_z.setPred_k$. Por tanto, es obvio que cualquiera de los nuevos mensajes ALG verifican la propiedad. En el caso de que $t \neq s_{z-1}.ta_i$, $m.ta \neq s_z.ta_i$, con lo que el antecedente de la propiedad en s_z no pasa a ser cierto. Para el resto de efectos de estas acciones, la hipótesis inductiva concluye.

- $\pi_z \in \{EndAddArc_j(i, t)\colon t \in \mathbb{N}\}$. Los mensajes ALG que pueden originarse cumplen que $m.ta = t$ e $(i, t) \in s_z.setPred_j$. La propiedad B.2.1 establece que $t \leq s_z.ta_i$. Si $t = s_z.ta_i$, basta aplicar la propiedad B.2.3 para concluir que el consecuente es cierto. Por otra parte, considerando que $t < s_z.ta_j$, la propiedad se cumple con el antecedente falso porque $m.ta < s_z.ta_i$.

- $\pi_z \in \{EndAddArc_x(i, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Si la acción está habilitada, entonces $(i, t, sent) \in s_{z-1}.set\_waiters_x$. Suponiendo que el antecedente es cierto, cuando el mensaje ALG se envió, el par $(i, s_{z-1}.ta_i) \in s_{z-1}.setPred_x$ o, equivalentemente por la propiedad B.3.2, $(i, s_{z-1}.ta_i, received) \in s_{z-1}.set\_waiters_x$. La presencia de esa tupla en $s_{z-1}.set\_waiters_x$ impide que cualquier otra tupla esté contenida en ese conjunto (propiedad B.1.5), se llega a una contradicción con la existencia de un mensaje ALG en s_{z-1}.

- $\pi_z \in \{StartDelArc_x(i, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. La acción no añade elementos al conjunto $setPred$ de ningún nodo, sólo los elimina. En consecuencia, el consecuente nunca podría llegar a ser falso en s_z, si en s_{z-1} fuera cierto.

- $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ o $\pi_z = Abort_i$. Aunque no crean ni eliminan mensajes de tipo ALG, los efectos de estas acciones modifican ta_i. De acuerdo con la propiedad B.2.2, un mensaje ALG que esté en el canal en s_{z-1} verifica que $m.ta \leq s_{z-1}.ta_i$. Tanto si el antecedente de la propiedad es falso como si es cierto en s_{z-1} por su característica temporal, el incremento del tiempo de activación del nodo i conlleva que $m.ta < s_z.ta_i$.

■

La existencia de un mensaje ALG en el canal con el tiempo correcto y que el nodo destino sea un predecesor directo del emisor del mensaje permite concluir, tal y

como señala la siguiente propiedad, que el nodo destino junto con su tiempo de activación también forman parte del conjunto de predecesores a los que se debe informar *setPredToInf* en el caso de que el nodo emisor rompa esa espera.

Propiedad B.2.41. Sea$\{i, j\} \subseteq \mathcal{N}$, se verifica que si $\exists\ m \in M_{ALG}$ tal que $m \in s.channel(j, i) \wedge m.ta = s.ta_i \wedge (i,s.ta_i) \in s.setPred_j \Rightarrow (i,s.ta_i) \in s.setPredToInf_j$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,j\} \subseteq \mathcal{N}$ se cumple que $s_0.channel(j, i) = \epsilon$ por lo que la propiedad es cierta.

- $\pi_z \in \{EndAddArc_j(i, t)\colon t \in \mathbb{N}\}$. Algunos efectos de la acción generan un mensaje ALG, m, con destino el nodo i y $m.ta = t$ tal que $(i, t) \in s_z.setPred_j$ y además $(i, t) \in s_z.setPredToInf_j$. Es obvio que la propiedad se verifica si t coincide con $s_z.ta_i$. Si $t \neq s_z.ta_i$, entonces la propiedad se cumple porque el antecedente es falso.

  Cuando se incorpora el par (i, t) en $s_z.setPred_j$, pero no se forma un mensaje ALG, hay que tener en cuenta la posibilidad de que en s_{z-1} pudiera existir un mensaje ALG que hiciera cierto el antecedente en s_z. Sabiendo que la acción queda habilitada si $(i, t, sent) \in s_{z-1}.set\_waiters_j$ y $s_{z-1}.status.alg_j \neq aborted$, se puede aplicar la propiedad B.1.5 para determinar que ni la tupla $(i, t, received)$ ni la tupla $(i, t, released)$ están incluidas en $s_{z-1}.set\_waiters_j$. En estas condiciones la propiedad B.2.38 asegura que no existe ningún mensaje ALG dirigido al nodo i con $m.ta = s_{z-1}.ta_i$. Dado que en el efecto analizado no se genera un mensaje ALG y la variable ta_i no cambia, se concluye que el antecedente de la propiedad es falso en esta situación.

- $\pi_z \in \{StartDelArc_j(i, t)\colon t \in \mathbb{N}\}$. Al ejecutarse la acción se elimina un elemento de $setPred_j$, pero nunca se añade al conjunto un nuevo par que pudiera hacer el antecedente cierto en s_z siendo falso en el estado previo.

- $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ o $\pi_z = Abort_i$. En la ejecución de ambas acciones se produce un incremento del tiempo de activación del nodo i, resultando $s_{z-1}.ta_i < s_z.ta_i$, pero no se generan ni eliminan mensajes ALG. Esto implica que, si existe un mensaje ALG hacia el nodo i verificando la propiedad en s_{z-1}, tras la ejecución de la acción resulta que $m.ta < s_z.ta_i$ haciendo falso el antecedente.

- $\pi_z = Abort_j$. Al ejecutarse esta acción se vacía tanto el conjunto $setPred_j$ como $setPredToInf_j$.

- $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$ o $\pi_z = initiate_j$ o $\pi_z \in \{rcvALG_j(x,m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ o $\pi_z \in \{rcvINF_j(x,m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$. Los efectos de estas acciones pueden llevar consigo la formación de un mensaje ALG dirigido

al nodo i tal que $m.ta = t$ y el par $(i,t) \in s_{z-1}.setPred_j$. De acuerdo con la propiedad B.2.1 $t \leq s_{z-1}.ta_i$ y sabiendo que los efectos de estas acciones no modifican ni ta_i ni $setPred_j$, se concluye que la propiedad es cierta. En el caso de que $t < s_{z-1}.ta_i$, con antecedente falso y si $t = s_{z-1}.ta_i$, con consecuente cierto porque el par $(i,\ t)$ también se añade a $s_z.setPredToInf_j$ al generarse el mensaje ALG.

■

Un nodo, que ha almacenado un mensaje ALG en su correspondiente *st_alg* y sigue manteniendo el bloqueo por su nodo sucesor, recibirá inforamación de este sucesor si se rompe la espera que los une. Para ello, la propiedad señala que el sucesor cuenta con su identidad y tiempo de activación en el conjunto *setPredToInf*.

Propiedad B.2.42. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$ que verifican que $s.st\_alg_i \neq NULL \wedge s.t\_unk_i = s.ta_i \wedge s.blocker_i = j \wedge (i,\ s.ta_i) \in s.setPred_j \Rightarrow (i,\ s.ta_i) \in s.setPredToInf_j$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ por lo que la propiedad es cierta. A continuación, se analizan los cambios de las variables de la propiedad por por efecto de las acciones.

- $\pi_z \in \{StartAddArc_i(x)\colon\ x \in \mathcal{N}\}$. Considerando que la acción esté habilitada, $s_{z-1}.state_i = active$ (según la propiedad B.3.3, $s_{z-1}.status.alg_i = active$). Si $s_{z-1}.status.id_i = known$, la propiedad B.2.13 señala que $s_{z-1}.st\_alg_i = NULL$. Si, por el contrario, $s_{z-1}.status.id_i = unknown$, la propiedad B.2.28 indica que $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$. Dado que los efectos de la acción no alteran ni la variable st_alg_i ni las variables t_unk_i y ta_i, se concluye que, en ambas situaciones, la propiedad se cumple con el antecedente falso.

- $\pi_z \in \{StartAddArc_j(x)\colon\ x \in \mathcal{N}\}$. En el supuesto de que al ejecutar la acción $s_z.status.alg_j$ se convierta en *candidate* y haya elementos que pertenecen al conjunto $s_{z-1}.setPred_j$ pero no a $s_{z-1}.setPredToInf_j$, se produce un efecto por el se incorporan a este último. Asumiendo que el tiempo de los pares que se incluyen son actuales, el consecuente de la propiedad pasa a ser cierto. Si los tiempos no se corresponde con los que los nodos presentan en ese instante, la propiedad también es cierta porque el antecedente es falso.

- $\pi_z \in \{EndAddArc_j(i,t)\colon\ t \in \mathbb{N}\}$. Si $s_{z-1}.status.alg_j \in \{dummy,\ candidate\}$,$(i,t) \in s_z.setPredToInf_j$. Es obvio que cuando $t = s_z.ta_i$ el consecuente de la propiedad es cierto en s_z. En el caso en el que el efecto de la acción consista sólo en añadir $(i,\ s_z.ta_i)$ a $s_z.setPred_j$ hay que comprobar que el antecedente de la propiedad en s_z no se convierta en cierto debido a ese efecto. La propiedad

B.2.39 establece que, si se verifica la primera parte del antecedente en s_{z-1}, se cumple que $(i,s_{z-1}.ta_i,received) \in s_{z-1}.set\_waiters_j$ o $(i,s_{z-1}.ta_i,released) \in s_{z-1}.set\_waiters_j$ o $s_{z-1}.status.alg_j = aborted$. Pero la condición de habilitación de la acción requiere que $(i,s_{z-1}.ta_i,sent) \in s_{z-1}.set\_waiters_j$ y $s_{z-1}.state_j \neq aborted$. Esto quiere decir que la acción no está habilitada.

- $\pi_z \in \{StartDelArc_i(x,\, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Entre los posibles efectos de la acción se encuentra uno en el que resulta $s_z.st\_alg_i = NULL$ y $s_z.t\_unk_i = -1$. En estas condiciones, el antecedente de la propiedad en s_z es falso.

- $\pi_z \in \{StartDelArc_j(i,\, t)\colon t \in \mathbb{N}\}$. Cuando se ejecuta esta acción, se elimina el par (i,t) de $s_{z-1}.setPred_j$. Si $t = s_z.ta_i$, el antecedente de la propiedad pasa a ser falso.

- $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$. En caso de que la acción esté habilitada, sus efectos hacen que el antecedente pase a ser falso porque $s_z.blocker_i = NULL$.

- $\pi_z = Abort_i$. Al ejecutarse la acción $s_z.status.alg_i = aborted$. La propiedad B.2.12 asegura que en ese estado $s_z.st\_alg_i = NULL$. Por consiguiente, la propiedad es cierta con el antecedente falso.

- $\pi_z = Abort_j$. Los efectos de la acción vacían tanto el conjunto $setPred_j$ como el conjunto $setPredToInf_j$.

- $\pi_z = initiate_i$. Si la acción está habilitada, $s_{z-1}.status.alg_i = blocked$ y además $s_{z-1}.status.id_i = known$. Según la propiedad B.2.13, $s_{z-1}.st\_alg_i = NULL$ en esas condiciones. Dado que los efectos de la acción no modifican st_alg_i, la propiedad se cumple con el antecedente falso.

- $\pi_z = initiate_j$. Cuando se ejecuta la acción, el contenido de de los conjuntos $setPredToInf_j$ y $setPred_j$ se iguala y el consecuente puede llegar a ser cierto si el tiempo de los elementos de los conjunto está actualizado.

- $\pi_z \in \{rcvALG_i(x,\, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$. Si los efectos de la acción provocan que $s_z.status.alg_i$ sea *victim*, la variable $st\_alg_i = NULL$. En esta situación, el antecedente de la propiedad sería falso. Sin embargo, cuando los efectos de la acción conllevan que $s_z.st\_alg_i \neq NULL$ y $s_z.t\_unk_i = s_z.ta_i$, se debe comprobar que la propiedad se verifica. Para ello, se analiza el mensaje ALG que se retira del canal por efecto de la acción. Si $(i,s_{z-1}.ta_i) \in s_{z-1}.setPred_x$, la propiedad B.2.41 indica que $(i,\, s_{z-1}.ta_i) \in s_{z-1}.setPredToInf_x$. Por lo tanto, resulta evidente que el consecuente es cierto en s_z porque la acción no modifica $setPredToInf_x$.

- $\pi_z \in \{rcvALG_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$. Si $s_z.status.alg_j = dummy$ tras ejecutar la acción, el contenido de $setPredtoInf_j$ puede cambiar. Todos los elementos del conjunto $setPred_j$ pasan a formar parte del conjunto $setPredToInf_j$. Siempre que los tiempos asociados a los nodos que incorporan estén actualizados, el consecuente de la propiedad será cierto.

- $\pi_z \in \{rcvINF_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Una vez ejecutada la acción $s_z.st\_alg_i = NULL$. Por tanto, la propiedad es cierta en s_z con el antecedente falso.

- $\pi_z \in \{rcvINF_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. En el caso de que, por efecto de la acción, $s_z.status.alg_j$ pase a ser *candidate*, los elementos de $s_{z-1}.setPred_j$ que no estén incluidos en $s_{z-1}.setPredToInf_j$ se añadirán a este último conjunto. De este modo, el consecuente de la propiedad podría ser cierto si el tiempo asociado a los nodos que se incorporan es el actual. La ejecución de la acción también puede eliminar pares de $s_{z-1}.setPredToInf_j$. Como los elementos que se retiran no están incluidos en $s_{z-1}.setPred_j$ y esta variable no cambia, el antecedente de la propiedad es falso tanto en s_{z-1} como en s_z.

- $\pi_z \in \{rcvAVS_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$. El efecto de la acción por el que $status\_alg_i$ pasa a ser *victim* va acompañado por otro efecto que hace que $s_z.st\_alg_i = NULL$.

■

Esta propiedad indica que los nodos bloqueados que conocen su identidad simulada, como no han participado de ningún proceso de detección, mantienen su conjunto *setPredToInf* vacío.

Propiedad B.2.43. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = blocked \wedge s.status.id_i = known \Rightarrow s.setPredToInf_i = \emptyset$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ por lo que la propiedad es cierta.

Considerando los efectos de las acciones que modifican la variable $status.alg_i$, sólo la acción $\pi_z \in \{StartAddArc_i(j)\colon j \in \mathcal{N}\}$ cambia a *blocked* esta variable. En caso de que $s_{z-1}.status.id_i$ sea *known* el antecedente podría llegar a ser cierto porque la acción no cambia el valor de $status.id_i$. Para comprobar que la propiedad es cierta en s_z, basta con observar que esta situación se produce cuando $s_{z-1}.setPredToInf_i = \emptyset$ y que este conjunto en s_z permanece vacío.

Respecto a la variable $setPredToInf_i$, hay que mencionar por un lado las acciones que añaden un elemento a ese conjunto, las que lo eliminan y las que lo vacían. Entre las acciones que incorporan un elemento a $setPredToInf_i$ se encuentran: $\pi_z \in$

$\{StartAddArc_i(j)\colon j \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = initiate_i$ (que puede añadir más de un elemento) y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$. En todos estos casos es evidente que el efecto comentado hace falso el consecuente, pero $s_z.status.alg_i \neq blocked$ lo que permite confirmar que la propiedad se cumple. Por otra parte, la ejecución de las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon j \in \mathcal{N} \wedge t \in \mathbb{N}\}$ y $\pi_z \in \{rcvINF_i(j,\ m)\colon j \in \mathcal{N} \wedge m \in M_{INF}\}$ puede llevar consigo la eliminación de un elemento de $setPredToInf_i$. Es posible que, al retirar el elemento del conjunto, $s_z.setPredToInf_i$ quede vacío, verificando así la propiedad. En cambio, si $s_z.setPredToInf_i \neq \emptyset$, la propiedad se verifica debido a que $s_z.status.alg_i \neq blocked$ y aunque $s_z.status.id_i = known$. Por último, hay que señalar que, al ejecutar la acción $\pi_z = Abort_i$, resulta que $s_z.setPredToInf_i = \emptyset$ y el consecuente de la propiedad pasa a ser cierto. ■

Básicamente esta propiedad indica que si dos mensajes ALG se dirigen a un mismo nodo, uno de ellos será obsoleto y corresponderá a una relación de espera que ya se ha roto y, por tanto, el tiempo de activación del nodo destino será inferior al que está vigente.

Propiedad B.2.44. Si $\exists\ \{i,\ j,\ k\} \subseteq \mathcal{N}$ y $\exists\ \{m,\ m\prime\} \subseteq M_{ALG}$ que verifican que $m \in s.channel(j,\ i) \wedge m\prime \in s.channel(k,\ i) \wedge m.ta = s.ta_i \Rightarrow m\prime.ta < s.ta_i$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,\ j\} \subseteq \in \mathcal{N}$ se cumple que $s_0.channel(i,\ j) = \epsilon$ por lo que la propiedad es cierta.

Suponiendo que existiera un mensaje ALG en s_{z-1}, $m\prime$, dirigido al nodo i procedente del nodo k, la propiedad B.2.40 establece que el par $(i,\ s_{z-1}.ta_i)$ no puede pertenecer al conjunto $setPred$ de ningún otro nodo que no sea el nodo k. Si se ejecutan las acciones que generan un nuevo mensaje ALG, m, que pudieran hacer el antecedente cierto en s_z, esto es, $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_j$, $\pi_z \in \{rcvALG_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$, se debe cumplir para todas ellas que $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPred_j$ y $m.ta = s_{z-1}.ta_i$. Dado que la ejecución de las acciones consideradas no modifica el conjunto $setPred$ de ningún nodo ni el tiempo de activación del nodo i, surge una contradicción porque el par $(i,\ s_z.ta_i)$ no puede estar incluido en el conjunto $setPred$ de dos nodos distintos (propiedad B.2.3). Tampoco puede suceder que el nodo k sea precisamente el nodo j que genera el nuevo mensaje ALG porque los mensajes no podrían tener el mismo tiempo asociado. Así que, la contradicción se deriva de la idea de partida de que los dos mensajes ALG puedan tener el mismo tiempo. De acuerdo a la propiedad B.2.2, $m\prime.ta \leq s_z.ta_i$ y en la situación descrita sólo es posible que $m\prime.ta < s_z.ta_i$ (consecuente cierto).

Cuando se ejecuta la acción $\pi_z \in \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$, también se forman un mensaje ALG siendo $s_{z-1}.status.alg_j \in \{blocked,\ candidate\}$. Para que la acción

se habilite, $(i,\ t,\ sent) \in s_{z-1}.set\_waiters_j$. Por otra parte, la propiedad B.1.5 indica que entonces, ni $(i,\ t,\ released)$ ni $(i,\ t,\ received)$ pertenecen a $s_{z-1}.set\_waiters_j$. Considerando que $t = s_{z-1}.ta_i$ y aplicando la propiedad B.2.38, se llega a la conclusión de que no existe ningún mensaje en el canal hacia el nodo i en s_{z-1}. Por consiguiente, el mensaje ALG generado por los efectos de la acción es el único que está en el canal y, por lo tanto, la propiedad se cumple con el antecedente falso.

En el caso de las acciones $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ y $\pi_z = Abort_i$, se sabe que sus efectos no crean ni eliminan mensajes de tipo ALG. Así que, para demostrar la propiedad se supondrá que en s_{z-1} existen los dos mensajes ALG dirigidos al nodo i. Dado que al ejecutarse la acción, el tiempo de activación del nodo i se incrementa, la propiedad se verificará con el antecedente falso porque $s_{z-1}.ta_i < s_z.ta_i$ y, en consecuencia, $m.ta < s_z.ta_i$.

■

En la siguiente propiedad se establece que un nodo que ya ha recibido un mensaje ALG, o que está a la espera de recibirlo, puede haber dejado de formar parte del conjunto de predecesores del nodo emisor o seguir en él.

Propiedad B.2.45. Sea $\{i,\ j\} \subseteq \mathcal{N}$, se verifica que si $s.t\_unk_i = s.ta_i \vee (\exists\ m \in M_{ALG}\colon m \in s.channel(j,\ i) \wedge m.ta = s.ta_i) \Rightarrow (i,\ s.ta_i) \notin s.setPred_j \vee (i,\ s.ta_i) \in s.setPredToInf_j$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.t\_unk_i = -1$, $s_0.ta_i = 1$ y $\forall\ \{i,\ j\} \subseteq \mathcal{N}$, $s_0.channel(j,\ i) = \epsilon$, por lo que la propiedad es cierta.

A continuación se van a analizar los efectos que podrían hacer que la propiedad pasase a ser falsa.

Considerando en primer lugar las acciones que generan mensajes de tipo ALG, se analiza que la propiedad se cumpla en s_{z-1}, o bien con el antecedente falso, o bien con el consecuente cierto.

- Si la propiedad se cumple en s_{z-1} con el antecedente falso y se ejecuta cualquiera de las acciones siguientes: $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$, $\pi_z = initiate_j$, $\pi_z \in \{rcvALG_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$, se puede formar un mensaje ALG, m, que haga el antecedente cierto. El nuevo mensaje ALG verifica que $m \in s_z.channel(j,\ i)$ y $m.ta = s_z.ta_i$. Además, al formarse el mensaje m, $(i,\ s_z.ta_i) \in s_z.setPredToInf_j$. Considerando todo ello, se confirma que la propiedad se cumple con el consecuente cierto en s_z.

- Si la propiedad se cumple en s_{z-1} con el consecuente cierto y se ejecutan las acciones anteriormente citadas surgen distintas posibilidades según qué condición hace cierto el antecedente:

1. $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$ y existe un mensaje ALG hacia el nodo i con $m.ta = s_{z-1}.ta_i$. El efecto de creación de un mensaje ALG dirigido al nodo i no puede producirse en ningún caso porque es necesario que $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPred_j \setminus s_{z-1}.setPredToInf_j$. En esas condiciones el consecuente sería falso y se suponía que era cierto en este análisis. Por tanto, el consecuente de la propiedad seguirá siendo cierto en s_z porque no se puede originar ningún nuevo mensaje ALG, $setPred_j$ no cambia y no se añaden nuevo elementos a $setPredToInf_j$.
2. $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$ y no existe en el canal ningún mensaje ALG dirigido al nodo i con $m.ta = s_{z-1}.ta_i$. Los efectos de las acciones no modifican ni t_unk_i ni ta_i. Por tanto, el antecedente de la propiedad sigue siendo cierto. Sin embargo, como es posible que se forme un mensaje ALG al ejecutarse cualquiera de estas acciones, hay que comprobar que el consecuente no pasa a ser falso. Si el mensaje ALG se crea cuando $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPred_j \setminus s_{z-1}.setPredToInf_j$ (acciones $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$ o $\pi_z \in \{rcvINF_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$), estas condiciones nunca podrán cumplirse porque se suponía que el consecuente era cierto en s_{z-1}. En conclusión, en este supuesto, el antecedente y el consecuente serán ciertos en s_z porque no es posible la formación de un mensaje ALG. En el caso de que el mensaje ALG se cree siendo $s_{z-1}.status.alg_j = blocked$ y $s_{z-1}.status.id_j = known$ (acciones $\pi_z = initiate_j$ o $\pi_z \in \{rcvALG_j(x,\ m\prime)\colon x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$), la propiedad B.2.43 establece que entonces $s_{z-1}.setPredToInf_j = \emptyset$. Para que el consecuente de la propiedad sea cierto en s_{z-1}, $(i,\ s_{z-1}.ta_i) \notin s_{z-1}.setPred_j$. Esto entra en contradicción con la condicón de generación del mensaje ALG. De nuevo, se concluye que la propiedad es cierta en s_z porque no pueden aparecer nuevos mensajes ALG y no cambia $setPred_j$.
3. $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$ y existe en el canal ningún mensaje ALG dirigido al nodo i con $m.ta = s_{z-1}.ta_i$. Para que $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$, se tuvo que ejecutar la acción $\pi_z \in \{rcvALG_i(x,\ m\prime\prime)\colon x \in \mathcal{N} \wedge m\prime\prime \in M_{ALG}$ en s_{z_1}, siendo $z_1 <$ *z-1*. Este mensaje ALG, $m\prime\prime$, se originó en s_{z_2} porque en ese instante $(i,\ t) \in s_{z_2}.setPred_x$. La recepción del mensaje $m\prime\prime$ en las condiciones del análisis implica que $t = s_{z_2}.ta_i = s_{z_1}.ta_i$. Cualquiera de las acciones que generan un mensaje ALG tienen como precocondición del envío que $(i,\ t) \in s_{z_2}.setPred_x$ y como efecto ligado al envío resulta que $(i,\ t) \in s_{z_2}.setPredToInf_x$. Si $x \neq j$ es necesario que se ejecuten las acciones $StartDelArc_x(i,\ t)$, $EndDelArc_i(x)$, $StartAddArc_i(j)$ y por último $EndAddArc_j(i,\ t\prime)$ para que $(i,\ t)$ pertenezca a $setPred_j$. Aunque de esta forma el nodo i logra estar en $setPred_j$, el tiempo asociado al nodo i en s_{z-1} ya no puede ser el mismo que aparecía en s_{z_2} porque la acción $EndDelArc_i(x)$

lo incrementa, $t < t\prime$. Si $x = j$, se tiene que $(i,\ t) \in s_{z_2}.setPred_j$ y $(i,\ t) \in s_{z_2}.setPredToInf_x$ pero, a cambio, se tiene que cumplir en s_{z-1} que $(i,\ t) \notin s_{z-1}.setPredToInf_j$ y $(i,\ t) \in s_{z-1}.setPred_x$. Como no se puede eliminar un elemento de $setPredToInf_j$ sin haberlo eliminado previamente de $setPred_j$, es necesario que se ejecute $StartDelArc_j(i,\ t)$, consiguiendo que $(i,\ t,\ released) \in s_{z_3}.set\_waiters_j$. Después se ejecutaría $EndDelArc_i(j)$ en s_{z_4} y sus efectos hacen que el tiempo de activación del nodo i se incremente,$t < t\prime$, y la variable $status.alg_i$ pase a ser *active*. En este estado la acción $StartAddArc_i(j)$ estará habilitada y, finalmente, tras la ejecución de $EndAddArc_j(i,\ t\prime)$, el nodo i pasará a ser un elemento de $s_{z-1}.setPred_j$. Sin embargo, el tiempo de activación con el que se incorpora a $setPred_j$ es mayor que el que se ha supuesto al considerar cierto el antecedente de la propiedad en s_{z-1}. Queda, por tanto, demostrado que el tiempo del mensaje ALG existente en s_{z-1} no puede coincidir con el tiempo t_unk_i del nodo al que va dirigido.

Por otra parte, las acciones que modifican la variable ta_i son: $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ y $\pi_z = Abort_i$. Al ejecutar cualquiera de estas acciones resulta que $s_{z-1}.ta_i < s_z.ta_i$. Por consiguiente, al ejecutarse la acción se obtiene que $s_z.t\_unk_i < s_z.ta_i$. Si el antecedente de la propiedad en s_{z-1} es cierto porque se verifica que $m.ta = s_{z-1}.ta_i$, al ejecutarse la acción, el antecedente pasará a ser falso. Cuando el antecedente de la propiedad es falso s_{z-1} porque $m.ta \neq s_{z-1}.ta_i$, concretamente $m.ta < s_{z-1}.ta_i$ según la propiedad B.2.2, el antecedente seguirá siendo falso en s_z ya que el incremento de ta_i hace que $m.ta < s_z.ta_i$.

Los efectos de las acciones $\pi_z \in StartDelArc_i(x,\ t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden hacer que $s_z.t\_unk_i$ adquiera el valor -1. Es evidente que entonces $s_z.t\_unk_i \neq s_z.ta_i$ ya que $s_z.ta_i \geq 1$ y el antecedente podría llegar a ser falso s_z.

Respecto a la variable $setPredToInf_j$, se puede observar que los efectos de las acciones: $\pi_z = \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z = \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$, $\pi_z = initiate_j$, $\pi_z = \{rcvALG_j(x,m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z = \{rcvINF_j(x,m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ pueden llegar a incluir el par $(i,\ s_z.ta_i)$ y, por tanto, hacer que el consecuente sea cierto. Por otro lado, el consecuente también puede convertirse en cierto cuando, al ejecutar las acciones $\pi_z = \{StartDelArc_j(i,\ t)\colon t \in \mathbb{N}\}$ y $\pi_z = Abort_j$, se elimina el par $(i,\ s_z.ta_i)$ de $setPred_j$. En el caso de la acción $\pi_z = Abort_j$, el conjunto $setPred_j$ se vacía completamente.

Hay que comprobar separadamente el efecto de la acción $\pi_z = \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$ que puede añadir el elemento $(i,\ s_z.ta_i)$ en $setPred_j$ sin incorporarlo a $setPredToInf_j$. Si, en esta situación, la propiedad se cumplía con el antecedente falso en s_{z-1}, el antecedente sigue siendo falso porque no se modifican las variables t_unk_i

y ta_i y no se genera un mensaje ALG. Además, el antecedente de la propiedad no podría ser cierto porque si se incorpora $(i,\ s_z.ta_i)$ en $s_z.setPred_j$, no es posible que el nodo i haya recibido o vaya a recibir un mensaje ALG con ese mismo tiempo.

De forma parecida, la acción $\pi_z = \{rcvINF_j(x,m)\colon\ x \in \mathcal{N} \wedge m \in M_{ALG}\}$ puede eliminar el par $(i,\ s_z.ta_i)$ de $setPredToInf_j$. Para que tenga lugar este efecto es necesario que el par eliminado no pertenezca a $s_{z-1}.setPred_j$. Como la ejecución de la acción no modifica el conjunto $setPred_j$ ni ta_i, el consecuente de la propiedad es cierto en s_z.

Finalmente, cuando se ejecuta la acción $\pi_z = \{rcvALG_i(j,m)\colon\ x \in \mathcal{N} \wedge m \in M_{ALG}\}$, se retira el mensaje ALG dirigido al nodo i que procede del nodo j. Así que, el antecedente podría ser falso y verificarse la propiedad.

■

En la siguiente propiedad se advierte que no puede haber un mensaje ALG en el canal dirigido a un nodo con tiempo correcto y que la variable t_unk del nodo destino tenga ya almacenado su tiempo de activación. Este hecho es imposible porque la variable t_unk asociada a un nodo almacena su valor tras la recepción del mensaje ALG y no antes.

Propiedad B.2.46. Si $\exists\ \{i,\ j\} \subseteq \mathcal{N}$ que verifican que $s.t\_unk_i = s.ta_i \Rightarrow (\nexists\ m \in m_{ALG}$ tal que $m \in s.channel(j,\ i) \wedge m.ta = s.ta_i)$.

Demostración: En el estado inicial, s_0, se cumple que $\forall\ \{i,\ j\} \subseteq \mathcal{N}\colon\ s_0.channel(j,\ i) = \epsilon$ por lo que la propiedad es cierta.

Al ejecutarse las acciones: $\pi_z \in \{StartDelArc_i(x,\ t)\colon\ x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{EndDelArc_i(x)\colon\ x \in \mathcal{N}\}$, $\pi_z \in \{rcvALG_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m \in M_{INF}\}$, es posible que $s_z.t\_unk_i = -1$. Como $s_z.t\_a_i > 0$, se deduce que $s_z.t\_unk_i \neq s_z.t\_a_i$. Por consiguiente, la propiedad se verifica con el antecedente falso.

Entre los efectos de las acciones $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$ y $\pi_z \in \{rcvINF_j(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m\prime \in M_{INF}\}$ se encuentra la formación de mensajes ALG. Los mensajes generados, m, con destino el nodo i verifican que $m.ta = t$ y el par $(i,\ t)$ pertenece a $s_{z-1}.setPred_j$. La propiedad B.2.45, siendo $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$, establece que $(i,\ s_{z-1}.ta_i) \notin s_{z-1}.setPred_j$ o $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPredToInf_j$. Esto implica que la condición de formación del mensaje ALG analizado, $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPred_j \setminus s_{z-1}.setPredToInf_j$ es incompatible con suponer que el antecedente de la propiedad es falso en s_{z-1}. Del mismo modo, se puede comprobar que si el mensaje ALG es generado por efecto de la acción $\pi_z = \{initiate_j\}$ o $\pi_z \in \{rcvALG_j(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m\prime \in M_{ALG}\}$, se requiere que $(i,\ t)$ pertenece a $s_{z-1}.setPred_j$. Aplicando la propiedad B.2.45, en los casos en que $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$, se asegura que o bien el par $(i,\ s_{z-1}.ta_i)$ es un elemento del conjunto $s_{z-1}.setPred_j$ o bien el par pertenece

a $s_{z-1}.setPredToInf_j$. La condición de habilitación de ambas acciones establece que $s_{z-1}.setPredToInf_j = \emptyset$ porque $s_{z-1}.status.alg_j = blocked$ y $s_{z-1}.status.id_j = known$ (propiedad B.2.43). Así que, no es posible que se cree el mensaje ALG si se considera que el antecedente es cierto en s_{z-1}.

Si el mensaje ALG se forma como efecto de la acción $\pi_z \in \{EndAddArc_j(i, t)\colon t \in \mathbb{N}\}$, $s_{z-1}.status.alg_j \in \{dummy, candidate\}$. Además la accón está habilitada cuando $(i, t, sent) \in s_{z-1}.set\_waiters_j$. Como la característica temporal del mensaje, $m.ta$, coincide con t, se hace evidente que en el caso de que t sea distinto a $s_{z-1}.ta_i$ la propiedad se verificará con el consecuente cierto. En cambio, considerando que $t = s_{z-1}.ta_i$ hay que confirmar que el antecedente de la propiedad es falso en s_{z-1} y que esa situación no cambia en s_z. La única acción que permite alcanzar la condición de habilitación es $StartAddArc_i(j)$. Los efectos de esta acción provocan que la variable $status.alg_i$ tome como valores *blocked* o *candidate*. En el primer estado, las propiedades B.2.28 y B.2.31, dependiendo del valor de $status.id_i$, señalan que $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$. Para el otro estado posible, *candidate*, la propiedad B.2.13 determina que $st\_alg_i = NULL$ y por la propiedad B.2.26, $s_{z-1}.t\_unk_i = -1$. Como la acción que se analiza no cambia las variables $s_{z-1}.t\_unk_i$ y $s_{z-1}.ta_i$, se puede concluir que la propiedad se verifica en cualquier circunstancia.

Por otra parte, resulta obvio que al retirarse el mensaje ALG por efecto de la acción $\pi_z \in \{rcvALG_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ se podría cumplir la propiedad con el consecuente cierto.

En lo que respecta a la variable ta_i, tanto la acción $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ como la acción $\pi_z = Abort_i$ incrementan el tiempo de activación del nodo i de manera que $s_{z-1}.ta_i < s_z.ta_i$. Como no se generan ni eliminan mensajes ALG al ejecutarse estas acciones, $m.ta < s_z.ta_i$ (propiedad B.2.2). Por tanto, el consecuente de la propiedad será cierto en s_z.

■

Según esta propiedad, toda ruta de más de un elemento que aparece durante la ejecución y cuyo último elemento representa una espera real en el sistema, también cuentan con información de esa espera en *setPredToInf* para poder corregir la información recogida si se suprimiera esa espera.

Propiedad B.2.47. Sea $p \in P^+$. Se verifica que $recordedPath(p, s) \wedge last(p) = (i, t_i) \wedge penult(p) = (j, t_j) \wedge (i, t_i) \in s.setPred_j \Rightarrow (i, t_i) \in s.setPredToInf_j$.

Demostración: En el estado inicial, s_0, se cumple que $\forall\ j \in \mathcal{N}\colon s_0.setPred_j = \emptyset$, por lo que la propiedad es cierta.

Las acciones que tienen como efecto añadir un elemento al conjunto $setPredToInf_j$ son: $\pi_z \in \{EndAddArc_j(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_j$, $\pi_z \in \{rcvALG_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_j(x, m)\colon x$

$\in \mathcal{N} \wedge m \in M_{INF}\}$. En todas ellas, el par que se incorpora está previamente en el conjunto $setPred_j$. Así que, en caso de que el par esté incluido en una de las rutas que se consideran, la propiedad se verificará con el antecedente y el consecuente ciertos.

De forma separada hay que estudiar un caso de ejecución de la acción $\pi_z \in \{EndAddArc_j(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ en el que se añade un par en $setPred_j$ pero no en $setPredToInf_j$. Según la condición de habilitación $(x, t, sent) \in s_{z-1}.set\_waiters_j$. Aplicando la propiedad B.1.5, se obtiene que $(x, t, received) \notin s_{z-1}.set\_waiters_j$. En esa situación, la propiedad B.3.2 establece que $(x, t) \notin s_{z-1}.setPred_j$. Como la propiedad se cumple en s_{z-1} por hipótesis inductiva, es obvio que lo hace con el antecedente falso. Si se produce el efecto indicado, el antecedente puede llegar a ser cierto siempre que en una ruta aparezcan el par (x, t) seguido del par (j, t_j). Dado que esta acción, en el caso mencionado, no incorpora elementos en ningún tipo de ruta, se concluye que la ruta se formó en un estado previo a su ejecución. Por consiguiente, el nodo j envió un mensaje ALG siendo (x, t) un par de $setPred_j$. Para alcanzar la condición de habilitación es necesario que el nodo x se active y vuelva a esperar de nuevo por el nodo j. En el proceso de activación, el tiempo asociado al nodo x se incrementa y, por tanto, el tiempo registrado en una ruta no coincidirá con el tiempo que se incorpora el nodo x en $setPred_j$. Según este razonamiento, si el antecedente de la propiedad es falso en s_{z-1}, también lo será en s_z.

Por el contrario, las acciones $\pi_z \in \{StartDelArc_j(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvINF_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z = Abort_j$ eliminan elementos del conjunto $setPredToInf_j$. En las dos primeras acciones, cuando se retira un par de $setPredToInf_j$, ese par ya no pertenece al conjunto $setPred_i$. En consecuencia, la acción se cumple con el antecedente y el consecuente falso. Además es posible que al ejecutarse la acción $\pi_z \in \{StartDelArc_j(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ sólo se elimine el par de $setPred_j$. En este caso, el antecedente de la propiedad podría convertirse en falso. Considerando la ejecución de la acción $\pi_z = Abort_j$, se observa que también se verifica la propiedad porque se vacían los conjuntos $setPred_j$ y $setPredToInf_j$.

■

Esta propiedad describe una característica importante de los mensajes AVS en retroceso. Un mensaje con el campo *fwd* a *true* ha dejado de incluir una ruta correcta porque se han borrado esperas del grafo. Se sabe con certeza que, al menos, el último nodo de la ruta se ha activado.

Propiedad B.2.48. Si $\exists\ \{i, j\} \subseteq \mathcal{N}$ que verifican que $\exists\ m \in M_{AVS}$ tal que $m \in s.channel(j, i) \wedge m.fwd = true \Rightarrow last(m.path) \notin s.setPred_j$.

Demostración: En el estado inicial, s_0, se cumple que $\forall\ \{i, j\} \subseteq \mathcal{N}\colon s_0.channel(j, i) = \epsilon$ por lo que la propiedad es cierta.

Las acciones $\pi_z \in \{StartDelArc_j(x,\ t)\colon\ x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_j$ y $\pi_z \in \{rcvAVS_j(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ permiten la generación de mensajes *AVS*, *m*, en los que *m.fwd* vale *true*. El destino de los mensaje viene dado en las dos primeras acciones por el identificador del penúltimo par de una ruta almacenada en $s_{z-1}.set\_st\_avs_j$ y en la última acción, por el identificador del penúltimo par de la ruta de $m\prime$. Como la ruta de los nuevos mensajes *AVS* verifica que $last(m.path)$ coincide con el penúltimo elemento de la ruta que sirve para su formación y es condición necesaria, en los tres casos, que el par $last(m.path)$ no esté contenido en el conjunto $setPred_j$, la propiedad queda demostrada.

Las acciones $\pi_z = firstAVS_j$ y $\pi_z = sndAVS_j$ también genera un mensaje *AVS* pero con el campo *fwd* a *false*. Así que, no afectan al cumplimiento de la propiedad. Por otro lado, el antecedente de la propiedad puede llegar a ser falso si se considera el efecto de la acción $\pi_z \in \{rcvAVS_i(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m\prime \in M_{AVS}\}$ por el que se retira del canal un mensaje *AVS*.

Considerando la variable $setPred_j$, se puede observar que al ejecutar las acciones $\pi_z = Abort_j$, $\pi_z \in \{StartDelArc_j(x,\ t)\colon\ x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ sólo eliminan elementos de *setPred*, por lo que pueden hacer cierto el consecuente. Como los efectos de esta acción no generan ni eliminan mensajes *AVS* del canal y tampoco se añaden elementos a $setPred_j$ que pudieran hacer el consecuente falso, se concluye que la propiedad es cierta.

Por el contrario, la acción $\pi_z \in \{EndAddArc_j(x,\ t)\colon\ x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ incorpora el par $(x,\ t)$ a $s_z.setPred_j$. Como los efectos de la acción no generan ni retiran del canal ningún mensaje *AVS*, el consecuente de la propiedad podría llegar a ser falso si $(x,\ t) = last(m.path)$, donde *m* es un mensaje *AVS* que cumple la propiedad en s_{z-1}. Para que esta acción esté habilitada es preciso que $(x,\ t,\ sent) \in s_{z-1}.set\_waiters_j$. Por la propiedad B.1.5, esa condición equivale a que $(x,\ t,\ received) \notin s_{z-1}.set\_waiters_j$ o $(x,\ t) \notin s_{z-1}.setPred_j$ según la propiedad B.3.2. Dado que $(x,\ t)$ debe aparecer como último elemento de la ruta del mensaje *AVS* en un estado s_{z_1} previo a s_{z-1} y que $(x,\ t) \notin s_{z-1}.setPred_j$, es necesario ejecutar la secuencia de acciones correspondiente para que el nodo *x* se active e inicie una espera por el nodo *j*. Al activarse el nodo *j*, su tiempo de activación se incrementa. Esto impide que el tiempo asociado al nodo *j* en la ruta del mensaje *AVS* coincida con el tiempo que quedará registrado cuando se complete la espera por el nodo *j*. Por consiguiente, la propiedad también se cumple al ejecutar esta acción.

■

Esta propiedad confirma que mientras en un canal haya un mensaje *ALG* con tiempo correcto dirigido a un nodo o un nodo tenga almacenada una ruta en su correspondiente *st_alg*, siendo *dummy* o *candidate*, el nodo estará bloqueado (existirá una espera parcial o total en el grafo del sistema). En el caso del mensaje, está bloqueado

por el emisor del mismo y si se trata del mensaje almacenado, el bloqueo es por el nodo que ocupa la primera posición de la ruta (*candidate*) o por el que ocupa la segunda posición (*dummy*).

Propiedad B.2.49. Sea $\{i, j\} \subseteq \mathcal{N}$. Si $\exists\ sid \in T, \exists\ p \in P^+$: $(ALG,\ s.ta_i,\ sid,\ p) \in s.channel(j,\ i) \vee (s.st\_alg_i = (sid,\ p) \wedge ((s.status.alg_i = dummy \wedge first(p - first(p)).id = j) \vee (s.status.alg_i = candidate \wedge first(p).id = j))) \Rightarrow s.blocker_i = j$.

Demostración: En el estado inicial, s_0, se cumple que $\forall\ \{i, j\} \subseteq \mathcal{N}$: $s_0.channel(j,\ i) = \epsilon$ y $s_0.st\_alg_i = NULL$, por lo que la propiedad es cierta. A continuación se analizan los efectos de las acciones que afectan a las variables de la propiedad.

Al ejecutarse las acciones $\pi_z \in \{StartAddArc_j(x)\colon x \in \mathcal{N}\}$, $\pi_z \in \{EndAddArc_j(i,\ t)\colon t \in \mathbb{N}\}$, $\pi_z = initiate_j$, $\pi_z \in \{rcvALG_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ puede generarse un mensaje de tipo ALG, m, que vaya dirigido al nodo i y cuyo $m.ta$ coincida con $s_z.ta_i$ o, según la propiedad B.3.1, $s_z.t\_activ_i$. Este efecto tiene lugar siempre que el par $(i,\ s_z.ta_i)$ esté contenido en $s_{z-1}.setPred_j$ o, lo que es lo mismo según la propiedad B.3.2, $(i,\ s_z.t\_activ_i,\ received) \in s_{z-1}.set\_waiters_j$. Por otra parte, la propiedad B.1.4 establece que, entonces, $s_{z-1}.blocker_i = j$. Como ninguna de las acciones modifica la variable ta_i ni la variable $blocker_i$, se concluye que la propiedad es cierta.

Obviamente, cuando se retira un mensaje ALG del canal, por efecto de la acción $\pi_z \in \{rcvALG_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$, el antecedente de la propiedad podría llegar a ser falso.

Las única acción que introduce la información requerida por el antecedente en st_alg_i es $\pi_z \in \{rcvALG_i(j,\ m)\colon m \in M_{ALG}\}$. Cuando se almacena el mensaje ALG siendo $s_{z-1}.status.alg_i = blocked$, resulta que $s_z.status.alg_i = dummy$ y $s_z.st\_alg_i.path = (i,\ s_z.ta_i).m.path$. Fácilmente se puede comprobar que $first(s_z.st\_alg_i.path - first(s_z.st\_alg_i.path)) = first(m.path)$. En el caso de que $s_{z-1}.status.alg_i = candidate$, los efectos de la acción mantienen el estado del nodo i y la ruta del mensaje ALG, $m.path$, se almacena tal cual en $s_z.st\_alg_i.path$. Es evidente que entonces, $first(s_z.st\_alg_i.path) = first(m.path)$. Tan sólo resta indicar que $first(m.path).id$ es justamente el nodo j (propiedad B.2.32). En ambos casos, por tanto, la existencia del mensaje ALG en s_{z-1} y la hipótesis inductiva, concluye.

Por otra parte, si se considera el efecto por el cual $s_z.st\_alg_i = NULL$, se deben analizar las acciones $\pi_z \in \{StartDelArc_i(x,\ t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{rcvALG_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ cuando $s_z.status.alg_i = victim$, $\pi_z \in \{rcvINF_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ y $\pi_z \in \{rcvAVS_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$ cuando $s_z.status.alg_i$ pasa a ser *victim*. En todos estas situaciones, el antecedente podría llegar a ser falso.

Otra variable a estudiar es $status.alg_i$. La única acción que permite alcanzar el estado *dummy* es $\pi_z \in \{rcvALG_i(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$. Este cambio de estado hace que el antecedente sea cierto como se acaba mencionar. Del mismo modo las

acciones $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_i$, $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ pueden convertir $status.alg_i$ en *candidate*. Salvo la acción $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ que viene acompañada de otro efecto que introduce información en $s_z.st\_alg_i$ y ya se ha comprobado que verifica la propiedad, el resto de acciones hacen que $s_z.st\_alg_i = NULL$. La condición de habilitación de las dos primeras acciones requiere que: $s_{z-1}.status.alg_i = blocked$ y $s_{z-1}.status.id_i = known$. La propiedad B.2.13 que en esa situación, $s_{z-1}.st\_alg_i = NULL$ y los efectos no modifican esta variable. Para la acción $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$ es evidente que $s_z.st\_alg_i = NULL$ porque es uno de los efectos de la propia acción.

Respecto a la variable $blocker_i$, la ejecución de la acción $\pi_z = StartAddArc_i(j)$ provoca que $s_z.blocker_i = j$ (consecuente cierto). En cambio, las acciones $\pi_z = Abort_i$ y $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ tienen como efecto $s_z.blocker_i = NULL$. Esto implica que para probar que la propiedad es cierta en s_z hay que demostrar que el antecedente es falso. Si se considera la variable st_alg_i, la condición de habilitación de la acción $\pi_z = Abort_i$ indica que $s_{z-1}.status.alg_i = victim$. En ese estado, de acuerdo con la propiedad B.2.12, $s_{z-1}.st\_alg_i = NULL$. Como esta variable no cambia su valor al ejecutar la acción, el antecedente sólo podría ser cierto si en s_z existe un mensaje ALG con destino el nodo i. En el caso de la acción $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$ puede suceder que $s_z.st\_alg_i = NULL$ o mantenga el contenido que tenía en s_{z-1}. Si $s_z.st\_alg_i \neq NULL$, dado que $s_z.status.alg_i = active$, la segunda parte del antecedente será definitivamente falsa. Por consiguiente, con esta acción sólo es posible que el antecedente sea cierto si en s_z existe un mensaje ALG dirigido al nodo i. Se sabe que los efectos de ambas acciones ni generan ni eliminan mensajes ALG, pero incrementan el tiempo de activación del nodo i de forma que $s_{z-1}.ta_i < s_z.ta_i$. Así que, si existe un mensaje ALG en s_{z-1}, tras la ejecución de cualquiera de las acciones, $m.ta < s_z.ta_i$ (propiedad B.2.2) y la propiedad se verificará con el antecedente falso.

■

En esta propiedad se establecen los posibles estados a los que puede evolucionar un nodo candidato que mantiene sus predecesores. Observando las transiciones de estado permitidas, se puede deducir que, en las condiciones indicadas, nunca podrá convertirse en *dummy*.

Propiedad B.2.50. Sea $\alpha = s_0\ \pi_1\ s_1 \ldots \pi_z\ s_z \ldots \in execs(S)$, una ejecución de S. Sea $\{i, j\} \subseteq \mathcal{N}$. Sea $t \in \mathbb{N}$. Se verifica que $\exists\ z$ tal que $s_z.status.alg_i = candidate \wedge (j, t) \in s_z.setPred_i$ entonces $\forall\ z\prime\colon z\prime > z$ si $(j, t) \in s_{z\prime}.setPred_i \Rightarrow s_{z\prime}.status.alg_i \in \{candidate, active, victim\} \vee (s_{z\prime}.status.alg_i = blocked \wedge s_{z\prime}.status.id_i = unknown)$.

Demostración: Se asume que en el estado s_z $s_z.status.alg_i = candidate$ y $(j, t) \in s_z.setPred_i$. Esta situación se ha podido alcanzar tras la ejecución de $\pi_z \in \{EndAddArc_i$

$(j,\ t)$: $t \in \mathbb{N}\}$ al incorporar $(j,\ t)$ en el conjunto $s_z.setPred_i$, siendo $s_{z-1}.status.alg_i = candidate$. Otras posibilidades son las acciones: $\pi_z \in \{StartAddArc_i(x)$: $x \in \mathcal{N}\}$, $\pi_z = initiate_i$ o $\pi_z \in \{rcvINF_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{INF}\}$. Estas acciones convierten $status.alg_i$ en $candidate$ siendo el par $(j,\ t)$ de $s_{z-1}.setPred_i$. Para la demostración de la propiedad se van a estudiar las posibles evoluciones a partir de este estado avanzando por una acción que modifica bien $setPred_i$ bien $status.alg_i$.

Las acciones que pueden estar habilitadas en s_z son:

- $\pi_{z\prime} \in \{EndAddArc_i(x,\ t)$: $x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Los efectos de esta ación provocan que un par $(x,\ t)$ se incorpore al conjunto $s_{z\prime}.setPred_i$. Como en ningún caso se elimina el par $(j,\ t)$ del conjunto y el estado del nodo i no varía ($s_{z\prime}.status.alg_i = candidate$), la propiedad se cumple.

- $\pi_{z\prime} \in \{StartDelArc_i(j,\ t)$: $t \in \mathbb{N}\}$. Al ejecutarse esta acción se mantiene $status.alg_i$, pero se retira el par $(j,\ t)$ de $setPred_i$. En estas circunstancias no se puede estudiar la propiedad.

- $\pi_{z\prime} \in \{EndDelArc_i(x)$: $x \in \mathcal{N}\}$. Con respecto a la ejecución de esta acción cuando $s_z.status.alg_i = candidate$, la propiedad B.2.11 permite confirmar que el par $(j,\ t)$ forma parte tanto de $s_z.setPred_i$ como de $s_z.setPredToInf_i$. Los efectos de la acción no modifican ninguno de estos conjuntos. Si, tras la ejecución de la acción, $s_{z\prime}.status.alg_i = active$ y $s_{z\prime}.status.id_i = known$, queda habilitada la acción $\{StartAddArc_i(x,\ t)$: $x \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Como $s_{z\prime}.setPredToInf_i \neq \emptyset$, si se ejecutara la acción, $status.alg_i$ volvería a ser $candidate$.

  En el caso de que $s_{z\prime}.status.alg_i = active$ y $s_{z\prime}.status.id_i = unknown$ es posible la habilitación de las acciones $\{StartAddArc_i(x,\ t)$: $x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\{rcvINF_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{INF}\}$. En la primera posibilidad, el nodo i cambia $status.alg_i$ a $blocked$, manteniendo el contenido de $setPred_i$ y $status.id_i$ como $unknown$. En esta nueva situación la propiedad se verifica o se permite la ejecución posterior de $\{rcvINF_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{INF}\}$. Los efectos de esta última provocan que $status.alg_i$ sea $candidate$ de nuevo porque $setPredToInf_i \neq \emptyset$. Dado que no se modifica $setPred_i$, es evidente que la propiedad se cumple. Si se ejecutara directamente la acción $\{rcvINF_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{INF}\}$, el estado del nodo i sería $active$-$known$ y $setPred_i$ permanecería sin cambios, cumpliéndose la propiedad.

- Cuando se ejecutan las acciones $\pi_{z\prime} \in \{rcvALG_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_{z\prime} \in \{rcvAVS_i(x,\ m)$: $x \in \mathcal{N} \wedge m \in M_{AVS}\}$ siendo $s_z.status.alg_i = candidate$, la propiedad se verifica ya que $s_{z\prime}.status.alg_i \in \{candidate,\ victim\}$. Si $s_{z\prime}.status.alg_i = victim$, sólo queda habilitada la acción $Abort_i$. Entre los posibles efectos de esta nueva acción destaca el que convierte $status.alg_i$ en $aborted$ y el que vacía

el conjunto $setPred_i$. Alcanzado el estado *aborted*, la propiedad B.1.6 asegura que $status.alg_i$ ya no variará. Como en $setPred_i$ no habrá ningún elemento, se dejan de cumplir las condiciones que marca la propiedad.

- Aunque las acciones $\pi_{z\prime} = firstAVS_i$, $\pi_{z\prime} \in \{dltINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$, $\pi_{z\prime} \in \{rcvAVSRSP_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$, $\pi_{z\prime} = sndAVS_i$ y $\pi_{z\prime} = sndAVSRSP_i$ están habilitadas, sus efectos no modifican ni la variable $setPred_i$ ni $status.alg_i$. La propiedad queda demostrada.

■

La siguiente propiedad sirve para caracterizar a los nodos candidatos que, al recibir el mensaje *ALG* de su candidato sucesor más próximo, respondieron con un mensaje *AVS* porque su identidad simulada era superior. Los candidatos que están en esta situación no tienen informción almacenada en su correspondiente *st_avsrsp*. Mientras el proceso de detección no avance y se localice a un candidato sucesor de mayor identidad simulada que él, este almacén permanecerá vacío.

Propiedad B.2.51. $\forall\ i \in \mathcal{N}$ se verifica que $s.status.alg_i = candidate \wedge s.nofirstAVS_i = false \wedge s.cand\_succ_i < s.sim\_id_i \Rightarrow s.st\_avsrsp_i = NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$, por lo que la propiedad es cierta. A continuación, se analizan los cambios que se producen en las variables de la propiedad al ejecutar las diferentes acciones.

Las acciones $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z = Abort_i$, $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$ pueden modificar el estado del nodo i de modo que $s_z.status.alg_i \neq candidate$. Este efecto convierte en falso el antecedente de la propiedad. En cambio, al ejecutarse las acciones $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_i$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$, es posible que $s_z.status.alg_i$ pase a ser *candidate*. Para que estén habilitadas estas acciones, $s_{z-1}.status.alg_i \in \{blocked, active\}$. La propiedad B.2.14 establece que en cualquiera de esos estados $s_{z-1}.st\_avsrsp_i = NULL$. Como los efectos de la acción no cambian la variable st_avsrsp_i, se puede afirmar que la propiedad es cierta en s_z.

Considerando la variable $nofirstAVS_i$, es obvio que la propiedad se verifica cuando se ejecuta la acción $\pi_z = firstAVS_i$ porque ésta adquiere el valor *false*. Las acciones en las que $nofirstAVS_i$ toma el valor *true* no se mencionan porque también modifican *status.alg* y ya han sido analizadas. Los efectos que modifican sim_id_i se localizan en las acciones $\pi_z \in \{StartDelArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Tras ejecutar la primera, $s_z.nofirstAVS_i = true$ y $s_z.sim\_id_i$ se reinicializa mientras $status.alg_i$ permanece siendo *active*. Según la propiedad B.2.14

a ese estado le corresponde que $s_z.st\_avsrsp_i = NULL$. Por otro lado, las dos últimas acciones ya han sido comprobadas al estudiar otras variables que intervienen en la propiedad.

Por otra parte, las únicas acciones que incluyen información en $s_z.st\_avsrsp_i$ son $\pi_z \in \{rcvALG_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVSRSP_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$. En el caso de la primera acción, el efecto se produce siendo $s_z.cand\_succ_i > s_z.sim\_id_i$. Esto implica que el antecedente de la propiedad es falso en s_z. Para comprobar que la propiedad se cumple al ejecutar la segunda acción hay que analizar el mensaje *AVSRSP*, m, que se retira del canal y se almacena en $s_z.st\_avsrsp_i$. De acuerdo con la propiedad B.2.18, el primer elemento de la ruta de ese mensaje es $(i,\ t)$ y además $m.ta$ coincide con t. Como el almacenamiento del mensaje tiene lugar cuando $m.ta = s_{z-1}.ta_i$, se puede aplicar la propiedad B.2.25 obteniéndose la siguiente relación, $m.sid > s_{z-1}.sim\_id_i$. Otro efecto que se produce en este caso consiste en que $s_z.cand\_succ_i$ adquiera el valor de $m.sid$. Sustituyendo estas variables en la desigualdad anterior resulta que $s_z.cand\_succ_i > s_z.sim\_id_i$ y el antecedente de la propiedad es falso.

La variable *cand_succ* puede adoptar el valor *NULL* tras la ejecución de las acciones $\pi_z \in \{EndDelArc_i(x)\colon\ x \in \mathcal{N}\}$, $\pi_z \in \{rcvALG_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(x,\ m)\colon\ x \in \mathcal{N} \wedge m \in M_{AVS}\}$. Este cambio en *cand_succ* no afecta al cumplimiento de la propiedad porque va acompañado de un cambio en *status.alg* ya mencionado. Sin embargo, conviene comentar que cualquier otro valor posible de *cand_succ* es superior a *NULL*, $\forall\ i \in \mathcal{N}$, $\forall\ t \in \mathbb{N}$ y $\forall\ \nu \in \mathbb{N}^*$ se considera que $(i,\ t,\ \nu) > NULL$. ■

Gracias a la siguiente propiedad se puede asegurar que la tercera componente del campo *sid*, que se transmite en un mensaje *INF*, es siempre distinta de la cadena vacía. De esta manera, se marca que el mensaje *INF* supera en su trayecto la primera o sucesivas esperas rotas del sistema.

Propiedad B.2.52. Sea $\{i,\ j\} \subseteq \mathcal{N}$. Si $\exists\ id \in \mathcal{N}$, $\exists\ \{ta,\ t\} \subseteq \mathbb{N}$, $\exists\ nu \in \mathbb{N}^*$: $(INF,\ (id,\ ta,\ nu),\ t) \in s.channel(i,\ j) \Rightarrow nu \neq \epsilon$.

Demostración: En el estado inicial, s_0, $\forall\ \{i,\ j\} \subseteq \mathcal{N}$ se cumple que $s_0.channel(i,\ j) = \epsilon$, por lo que la propiedad es cierta.

Las acciones $\pi_z \in \{StartDelArc_i(j,\ t)\colon\ t \in \mathbb{N}\}$ y $\pi_z = Abort_i$ generan un mensaje de tipo *INF*. El campo *sid* de este mensaje es construido a partir del valor de $s_{z-1}.sim\_id_i$, esto es, $(s_{z-1}.sim\_id_i.id,\ s_{z-1}.sim\_id_i.ta,\ s_{z-1}.sim\_id_i.nu\ .\ s_{z-1}.cont_i)$. Como la variable $cont_i$ inicialmente tiene el valor 1 y se incrementa en una unidad para cada mensaje *INF* que el nodo i envía, resulta evidente que el valor correspondiente a nu es distinto a ϵ y la variable $cont_i$ siempre cuenta con un valor. Del mismo modo, si se forman nuevos mensajes *INF* por efecto de la acción $\pi_z \in \{rcvINF_i(x,\ m\prime)\colon\ x \in \mathcal{N} \wedge m\prime$

$\in M_{INF}\}$, el campo *nu* de la variable *sid* que incluyen los mensajes consiste en la concatenación de $s_z.sim\_id_i.nu$ y $s_z.cont_i$. Como la variable $cont_i$ siempre dispone de valor $nu \neq \epsilon$.

Por otra parte, al ejecutarse las acciones $\pi_z \in \{dltINF_j(i, m)\colon m \in M_{INF}\}$ y $\pi_z \in \{rcvINF_j(i, m)\colon m \in M_{INF}\}$ se produce como efecto la retirada del mensaje *INF* que ocupaba el canal en s_{z-1}. Es evidente que este efecto puede hacer falso el antecedente de la propiedad.

■

De acuerdo a esta propiedad, se puede afirmar que los nodos que tienen información en su *setPredToInf* y su identidad simulada coincide con un valor inicializado son nodos bloqueados en el medio.

Propiedad B.2.53. Sea $i \in \mathcal{N}$. Si $s.setPredToInf_i \neq \emptyset \wedge s.sim\_id_i = (i, s.ta_i, \epsilon) \Rightarrow s.state_i = blocked$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.setPredToInf_i = \emptyset$ por lo que la propiedad es cierta.

Al ejecutarse las siguientes acciones $\pi_z \in \{StartAddArc_i(x)\colon x \in \mathcal{N}\}$, $\pi_z = initiate_i$, $\pi_z \in \{rcvALG_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$, $\pi_z \in \{rcvAVS_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$, se puede modificar la variable $status_i$ de manera que $s_z.status.alg_i \in \{blocked, dummy, candidate, victim\}$. Dados cualquiera de esos estados, la propiedad B.3.4 establece que $s_z.state_i = blocked$. Así que, la propiedad se verifica con el consecuente cierto.

En lo que respecta al conjunto $setPredToInf_i$, la acción $\pi_z = Abort_i$ elimina todos sus elementos. Por tanto, los efectos de esta acción convierten en falso el antecedente de la propiedad.

Si se ejecuta la acción $\pi_z \in \{EndAddArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ es posible que se incorpore el par (x, t) al conjunto $s_z.setPredToInf_i$. Este efecto se produce sólo cuando $s_{z-1}.status.alg_i \in \{dummy, candidate\}$. Como la variable $status.alg_i$ no varía y la propiedad B.3.4 indica que esos estados equivalen a $s_z.state_i = blocked$, se concluye que, en este supuesto, la propiedad se verifica con el consecuente cierto.

Las acciones que pueden eliminar elementos del conjunto $setPredToInf_i$ son $\pi_z \in \{StartDelArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Si, tras ejecutarse estas acciones, $s_z.setPredToInf_i = \emptyset$, es obvio que la propiedad se cumple con el antecedente falso. En cambio, en el caso de que al eliminar un par de $setPredToInf_i$ resulte $s_z.setPredToInf_i \neq \emptyset$, se podrá aplicar la hipótesis inductiva para demostrar la propiedad siempre que el resto de variables no cambien su valor.

Los efectos que modifican la variable sim_id_i se producen al ejecutar las acciones $\pi_z \in \{StartDelArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z = Abort_i$ y $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. En los tres casos, $s_z.sim\_id_i$ adquiere el valor $(i, s_z.ta_i, \epsilon)$ siendo

$s_z.setPredToInf_i = \emptyset$ (antecedente falso). Para confirmar que $s_z.setPredToInf_i = \emptyset$ en el efecto comentado de $\pi_z \in \{StartDelArc_i(x, t)\colon x \in \mathcal{N} \wedge t \in \mathbb{N}\}$ es preciso considerar la propiedad B.2.4 que asegura que $s_z.setPredToInf_i$ está vacío, si $s_z.setPred_i = \emptyset$ y $s_z.status.id_i = known$. Por otra parte, es posible que $s_z.sim\_id_i$ adopte otro valor, diferente a $s_{z-1}.sim\_id_i$, por efecto de la acción $\pi_z \in \{rcvINF_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Si $s_z.sim\_id_i = m.sid$ siendo m el mensaje INF que estaba en el canal en s_{z-1}, la propiedad queda demostrada porque la propiedad B.2.52 asegura que $m.sid.nu \neq \epsilon$.

Por último, si se analiza el cambio en la variable ta_i cuando se ejecuta la acción $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$, resulta evidente que el antecedente de la propiedad pasa a ser falso porque el tiempo de activación del nodo i se incrementa ($s_{z-1}.ta_i < s_z.ta_i$), pero sim_id_i no se actualiza y $s_z.sim\_id_i \neq (i, s_z.ta_i, \epsilon)$.

■

Esta propiedad recoge todos las posibilidades de evolución en el sistema y cómo afecta a la recepción y al almacenamiento de los mensajes ALG. Además, se asegura que la identidad simulada del nodo candidato mayor conocido mediante este tipo de mensajes se preserva. Si ese nodo candidato se activa, el resto de nodos que le preceden podrán enterarse de la situación y adaptarse a los cambios necesarios para que el proceso de detección pueda continuar. Para ello, es necesario disponer de un camino de esperas borradas, *ewait* (definición 6.1.1 del capítulo 6).

Propiedad B.2.54. Sea $\{i, j, k, n\} \subseteq \mathcal{N}$, sea $\{sid, sid\prime\} \in T$ y sea $\{p, p\prime\} \in P$. Se verifican simultáneamente las aserciones *A1*, *A2* y *A3*.
A1: $(ALG, s.ta_i, sid, p) \in s.channel(j, i) \Rightarrow$

- $(s.status.id_j = known \wedge$
  - *C1-1*: $((sid \leq s.sim\_id_j \wedge (i, s.ta_i) \in s.setPredToInf_j) \vee$
  - *C1-2*: $(s.status.alg_j = dummy \wedge (i, s.ta_i) \in s.setPredToInf_j) \vee$
  - *C1-3*: $((INF, sid\prime, s.ta_i) \in s.channel(j, i) \wedge sid\prime > sid)))$

  $\vee$
- $(s.status.id_j = unknown \wedge ((i, s.ta_i, received) \in s.set\_waiters_j \vee (i, s.ta_i, released) \in s.set\_waiters_j) \wedge$
  - *C1-4*: $(((i, s.ta_i) \in s.setPredToInf_j \wedge (INF, sid\prime, s.t\_unk_j) \in s.channel(k, j) \wedge ((s.inf\_need_j = true \wedge s.sim\_id_j \geq sid) \vee (s.inf\_need_j = false \wedge sid\prime > sid))) \vee$

- *C1-5*: $((i,\ s.ta_i) \in s.setPredToInf_j \land ewait(j,\ last(p\prime).id,\ p\prime,\ s) \land (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,\ last(p\prime).id) \land$
  - $((last(p).id \in nodes(p\prime) \land s_z.sim\_id_{last(p).id} \geq sid \land \forall\ (x,\ t_x) \in (visited\_nodes(p)\text{-}last(p))$: $s.inf\_need_x = false \land s.inf\_need_{last(p).id} = true)$ $\lor$
  - $(last(p).id \notin nodes(p\prime) \land sid\prime > sid \land \forall\ (x,\ t_x) \in visited\_nodes(p\prime)$: $s.inf\_need_x = false))))$

A2: $s.st\_alg_i = (sid,\ p) \land s.t\_unk_i = s.ta_i \land s.blocker_i = j \Rightarrow$

- $(s.status.id_j = known \land$
  - *C2-1*: $(((sid \leq s.sim\_id_j \land (i,\ s.ta_i) \in s.setPredToInf_j) \lor$
  - *C2-2*: $(s.status.alg_j = dummy \land (i,\ s.ta_i) \in s.setPredToInf_j) \lor$
  - *C2-3*: $((INF,\ sid\prime,\ s.t\_unk_i) \in s.channel(j,\ i) \land sid\prime > sid)))$

  $\lor$

- $(s.status.id_j = unknown \land$
  - *C2-4*: $((i,\ s.ta_i,\ received) \in s.set\_waiters_j \land$
    - *C2-4a*: $((ewait(j,\ last(p\prime).id,\ p\prime,\ s) \land (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,\ last(p\prime).id) \land$
      - $((last(p).id \in nodes(p\prime) \land s.sim\_id_{last(p).id} \geq sid \land \forall\ (x,\ t_x) \in (visited\_nodes(p) - last(p))$: $s.inf\_need_x = false \land s.inf\_need_{last(p).id} = true) \lor$
      - $(last(p).id \notin nodes(p\prime) \land sid\prime > sid \land \forall\ (x,\ t_x) \in visited\_nodes(p\prime)$: $s.inf\_need_x = false))$

      $\lor$
    - *C2-4b*: $((INF,\ sid\prime,\ s.t\_unk_j) \in s.channel(k,\ j) \land ((s.inf\_need_j = true \land s.sim\_id_j \geq sid) \lor (s.inf\_need_j = false \land sid\prime > sid))))))$

    $\lor$
  - *C2-5*: $((i,\ s.ta_i,\ released) \in s.set\_waiters_j \land ewait(i,\ last(p\prime).id,\ p\prime,\ s) \land (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,\ last(p\prime).id) \land$
    - $((last(p).id \in nodes(p\prime) \land s.sim\_id_{last(p).id} \geq sid \land \forall\ (x,\ t_x) \in (visited\_nodes(p) - last(p) - first(p\prime))$: $s.inf\_need_x = false \land s.inf\_need_{last(p).id} = true) \lor$
    - $(last(p).id \notin nodes(p\prime) \land sid\prime > sid \land \forall\ (x,\ t_x) \in (visited\_nodes(p\prime) - first(p\prime))$: $s.inf\_need_x = false))))$

A3: $s.st\_alg_i = (sid,\ p) \wedge s.t\_unk_i \neq s.ta_i \Rightarrow$

- *C3-1*: $(ewait(i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(n,\ last(p\prime).id) \wedge$

  - $((last(p).id \in nodes(p\prime) \wedge s.sim\_id_{last(p).id} \geq sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p) - last(p) - first(p\prime))$: $s.inf\_need_x = false \wedge s.inf\_need_{last(p).id} = true) \vee$
  - $(last(p).id \notin nodes(p\prime) \wedge sid\prime > sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p\prime) - first(p\prime))$: $s.inf\_need_x = false)))$

  $\vee$

- *C3-2*: $((INF,\ sid\prime,\ s.t\_unk_i) \in s.channel(j,\ i) \wedge sid\prime > sid)$

Demostración: En el estado inicial, s_0, $\forall\ \{i,\ j\} \subseteq \mathcal{N}$ se cumple que $s_0.st\_alg_i = NULL$ y $s_0.channel(i,\ j) = \epsilon$ por lo que viola el antecedente de las tres aserciones.

- $\pi_z \in \{StartAddArc_j(x)$: $x \in \mathcal{N}\}$. Cuando $s_{z-1}.status.id_j = known$ es posible que se genere de un mensaje ALG con destino al nodo i, m, que haga cierto el antecedente $(A1)$. Como al mismo tiempo se incorpora el par $(i,\ s_z.ta_i)$ en el conjunto $s_z.setPredToInf_j$ y el campo sid del mensaje coincide con $s_z.sim\_id_j$, el consecuente de la propiedad $(C1\text{-}1)$ también es cierto al considerar este efecto.

- $\pi_z = StartAddArc_i(j)$. Esta acción no puede hacer cierto ni el antecedente de *A1* ni el de *A3* ni tampoco modifica ninguna de las variables asociadas al nodo i, que aparecen en sus distintos consecuentes. Al ejecutarse esta acción, la tupla $(i,\ s_z.ta_i,\ sent)$ se incorpora a $s_z.set\_waiters_j$.

  Por otra parte, el antecedente *A2* de la propiedad podría convertirse en cierto siendo falso en s_{z-1} ya que $s_z.blocker_i = j$. Para que esto sea posible es preciso que $s_{z-1}.st\_alg_i \neq NULL$ y $s_{z-1}.t\_unk_i = s_{z-1}.ta_i$. Cuando la acción está habilitada $s_{z-1}.status.alg_i = active$. Si $s_{z-1}.status.id_i = known$, la propiedad B.2.13 establece que $s_{z-1}.st\_alg_i = NULL$. Dado que los efectos de la acción en este caso no modifican esta variable, el antecedente de la propiedad será falso en s_{z-1} aunque se añada el nodo j como $blocker_i$. Del mismo modo, si $s_{z-1}.status.id_i = unknown$ el antecedente de la propiedad es falso en s_{z-1} según la propiedad B.2.31 porque $s_{z-1}.t\_unk_i \neq s_{z-1}.ta_i$. Esta variables no cambian al ejecutar la acción y el antecedente de la propiedad será falso en s_z aunque $blocker_i$ pase a ser el nodo j.

- $\pi_z \in \{EndAddArc_j(i,\ t)$: $t \in \mathbb{N}\}$. Uno de los posibles efectos de esta acción consiste en la formación de un mensaje ALG, m, que cumple que $m \in s_z.channel(j,\ i)$ y $m.ta = s_z.ta_i$. Si $s_{z-1}.status.alg_j = dummy$, la ejecución de la acción no

modifica la variable $status.alg_j$ y el consecuente *C1-2* se hace cierto porque $(i,\ s_z.ta_i) \in s_z.setPredToInf_j$. Cuando $s_{z-1}.status.alg_j = candidate$ se verifica que $m.sid = s_{z-1}.sim\_id_j$ y $(i,\ s_z.ta_i) \in s_z.setPredToInf_j$. Como $s_{z-1}.sim\_id_j = s_z.sim\_id_j$ es obvio que la propiedad se cumple con el consecuente *C1-1* cierto. Si en s_{z-1} el antecedente *A2* era verdadero, por hipótesis inductiva el consecuente *C2-5* tenía que ser verdadero y $(i,\ s_{z-1}.ta_i,\ released)$. Esto impide que la acción esté habilitada.

- $\pi_z \in \{EndAddArc_i(j,\ t)\colon t \in \mathbb{N}\}$. Aunque las variables relacionadas con el nodo i que aparecen en el antecedente de la propiedad no se ven modificadas, se debe comprobar que la estructura tanto de $ewait(i,\ last(p\prime),\ p\prime,\ s_z)$ como de $ewait(j,\ last(p\prime),\ p\prime,\ s_z)$ sigue cumpliendo la definición tras la ejecución de la acción. Considerando $ewait(j,\ last(p\prime),\ p\prime,\ s_{z-1})$, si $(i,\ s_z.ta_i)$ se incluye en $s_z.setPred_j$

- $\pi_z \in \{StartDelArc_i(j,\ t)\colon t \in \mathbb{N}\}$. Suponiendo que $s_{z-1}.status.alg_j = aborted$, los efectos de la acción no modifican el valor de ninguna de las variables que participan en la propiedad. Aplicando la hipótesis inductiva, se concluye. Por otro lado, en el supuesto de que $s_{z-1}.status.alg_i = active$, como los efectos de la acción no introducen cambios en ninguna variable asociada al nodo j y no se retiran los mensajes ALG o INF que pudieran tener como destino el nodo i, la hipótesis inductiva concluye. Entre los posibles efectos de la acción está el que hace que $s_z.t\_unk_i$ pase a valer -1 a la vez que se vacía $s_z.st\_alg_i$. Obviamente estos cambios pueden convertir en falsos los antecedentes de la propiedad.

- $\pi_z \in \{StartDelArc_j(i,\ t)\colon t \in \mathbb{N}\}$. Cuando la acción queda habilitada siendo $s_{z-1}.status.alg_i = aborted$, la propiedad en s_{z-1} se verifica con el antecedente falso. Según la propiedad B.2.12, $s_{z-1}.st\_alg_i = NULL$ y, la propiedad B.2.8 confirma que el tiempo asociado a un mensaje ALG dirigido al nodo i es inferior a $s_{z-1}.ta_i$. Después de ejecutarse la acción, la propiedad sigue verificándose con el antecedente falso porque no hay cambios en las variables y mensajes citados.

  Si $s_{z-1}.status.alg_j = active$ y los antecedentes de la propiedad son falsos en s_{z-1}, la propiedad sigue siendo cierta porque las variables del antecedente no varían a excepción de $set\_waiters_j$ que elimina la tupla $(i,\ s_{z-1}.ta_i,\ received)$. Este cambio no influye porque en lugar de esta tupla se incorpora la tupla $(i,\ s_z.ta_i,\ released)$ que puede aparecer igualmente en el antecedente en s_{z-1}. Además, los efectos de la acción no generan ni retiran mensajes de tipo ALG dirigidos al nodo i.

  Por las mismas razones, la ejecución de la acción mantiene como ciertos los antecedentes de la propiedad si lo eran en s_{z-1}. Sin embargo, en esta situación deben analizarse las variables de los posibles consecuentes para comprobar que alguno de ellos sigue siendo cierto tras ejecutarse la acción.

Cuando $s_{z-1}.status.id_j = known$ se pueden verificar tanto los consecuentes *C1-1* y *C1-3* asociados al antecedente *A1* como *C2-1* y *C2-3* asociados al antecedente *A2*. Los efectos de la acción hacen que *C1-1* o *C3-1* pasen a ser falsos debido a que el par $(i,\ s_{z-1}.ta_i)$ se suprime del conjunto $setPredToInf_j$, pero al mismo tiempo se forma un mensaje *INF* del nodo j al nodo i que verifica *C1-3* o *C2-3* respectivamente. En ambos casos la característica temporal del mensaje *INF* es correcta porque las variables ta_i y t_unk_i no cambian. Por otra parte, el campo *sid'* es mayor que el correspondiente campo *sid* del mensaje *ALG* del antecedente *A1* o del mensaje almacenado en st_alg_i (antecedente *A2*). Para verificarlo basta con recordar que $s_{s-1}.sim\_id_j$ era superior a ellos y finalmente, *sid'* se construye a partir de $s_{s-1}.sim\_id_j$ cumpliendo la relación de orden. Es fácil comprobar que *C1-3* y *C2-3* seguirán siendo ciertos en s_z porque el mensaje *INF* existente en el canal y el resto de variables incluidas en ellos no sufren variaciones.

En el caso que $s_{z-1}.status.id_j = unknown$, se observa que los consecuentes posibles para los antecedentes *A1* (*C1-4* y *C1-5*) y *A3* (*C3-1* y *C3-2*), al ejecutarse la acción, siguen verificándose. En este supuesto de ejecución, las variables que conforman estos consecuentes no cambian, los mensajes *INF* no se retiran del canal y mantienen sus características y las rutas tipo *ewait* cumplen la definición. Aunque la habilitación de la acción requiere que el par $(i,\ s_{z-1}.ta_i)$ esté contenido en $s_{z-1}.setPred_j$, la existencia de $ewait(i,\ last(p'),\ p',\ s_{z-1})$, como indica *C3-1*, supone que el nodo i se desactivó y volvió a bloquearse con un tiempo de activación distinto al registrado en dicho *ewait* antes de que la acción pueda ejecutarse. Esto implica que la definición de este *ewait* se verifica tras ejecutar la acción.

Considerando cierto el antecedente *A2* en s_{z-1} y aplicando la hipótesis inductiva, se deduce que en s_{z-1} se cumple el consecuente *C2-4a* o *C2-4b* porque son los únicos que satisfacen la condición de habilitación, $(i,\ s_{z-1}.ta_i,\ received) \in s_{z-1}.set\_waiters_j$. Para que la propiedad se cumpla es preciso, en ambos casos, que se cumpla el consecuente *C2-5* ya que los efectos de la acción hacen que $(i,\ s_z.ta_i,\ released)$ pase a formar parte de $s_z.set\_waiters_j$ y conserva los valores de ta_i y $status.id_j$. El mensaje *INF* del consecuente *C2-5* es exactamente el que incluye el consecuente *C2-4a*, pero el camino *ewait* se amplía y pasa a tener como primer elemento el nodo i por efectos de la acción. Aunque el par $(i,\ s_z.ta_i)$ deja de pertenecer al conjunto $setPred_j$ sigue perteneciendo al conjunto $setPredToInf_j$. La propiedad B.2.42 asegura que, en las condiciones del antecedente *A2*, $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPredToInf_j$. Dado que los efectos de la acción no modifican la variable t_unk_i, el campo *sid* almacenado en st_alg_i y tanto las variables *inf_need* como *sim_id* de los nodos que forman parte de las rutas p y

$p\prime$ en s_{z-1}. Por todo ello, se cumple la definición $ewait(i,\ last(p\prime),\ p\prime,\ s_z)$ y el consecuente *C2-5* en s_z.

Observando el consecuente *C2-4b* se aprecia que el mensaje *INF* existente en s_{z-1} viaja del nodo k al nodo j. Cuando se ejecuta la acción, se forma un camino $ewait\ (i,\ last(p\prime),\ p\prime,\ s_z)$ similar al descrito para el consecuente *C2-4a*. La ruta $p\prime$ sólo contiene al nodo i y al nodo j. Por otra parte, el mensaje *INF* sigue en el canal en s_z, pero empleando las referencias de ruta $p\prime$, su origen es el nodo n (antes nodo k) y su destino es $last(p\prime)$, o sea, el nodo j. De acuerdo con el valor de $s_{z-1}.inf\_need_j$ se cumple características diferentes que recoge el consecuente *C2-5*.

Si $s_{z-1}.inf\_need_j = true$, el nodo j, antes de que $s_{z-1}.status.id_j = unknown$, era *candidate*. En consecuencia, la ruta p almacenada en $s_{z-1}.st\_alg_i$ se corresponde prácticamente con el mensaje ALG que envió el nodo j y $last(p).id = j$. Esto confirma que $last(p).id \in nodes(p\prime)$ manteniéndose, al ejecutar la acción,$s_z.sim\_id_j \geq sid$. También se cumple que todos los nodos de p excepto el último nodo de p (nodo j) y el primero de $p\prime$ (nodo i) Como $p\prime$ tiene precisamente sólo esos dos nodos, no se puede conocer el valor de *inf_need* para ningún otro nodo. Como inf_need_j no varía, $s_z.inf\_need_j = true$, confirmando que el consecuente *C2-5* en s_z es cierto en este caso.

Cuando $s_{z-1}.inf\_need_j = false$, se sabe que el nodo j era *dummy* antes de que $s_{z-1}.status.id_j = unknown$. En esa situación se deduce que las rutas almacenadas en $s_{z-1}.st\_alg_i$ y en $s_{z-1}.st\_alg_j$ coinciden prácticamente en su totalidad y obligatoriamente en su último par. Si $last(p).id \notin nodes(p\prime)$ entonces $last(s_{z-1}.st\_alg_j.path).id \notin nodes(p\prime)$ y, por tanto, el nodo que mandó el mensaje ALG almacenado no está contenido en $p\prime$. Esto implica que $sid\prime > sid$ en s_z porque $s_{z-1}.st\_alg_i = s_{z-1}.st\_alg_i$. Además, de los nodos que forman parte de $p\prime$ a excepción del primero, se conoce el valor de su *inf_need*. En este caso sólo del nodo j que es $last(p\prime.id)$ y que no ha variado al ejecutar la acción, siendo $s_z.inf\_need_j = false$. De esta forma, se concluye que el consecuente *C2-5* se verifica en s_z.

- $\pi_z \in \{EndDelArc_i(x)\colon x \in \mathcal{N}\}$. Si el antecedente *A1* era cierto en s_{z-1}, al ejecutarse esta acción se incrementa el tiempo de activación del nodo i, la condición temporal del mensaje ALG deja de cumplirse porque $m.ta < s_z.ta_i$ y el antecedente pasa a ser falso. Por otra parte, cuando el antecedente *A2* de la propiedad se verificaba en s_{z-1}, el efecto de la acción que hace que $s_z.blocker_i = NULL$ convierte en falso el antecedente. El antecedente *A2* también podría llegar a ser falso al eliminarse el elemento $(i,\ s_{z-1}.ta_i,\ released)$ del conjunto $set\_waiters_j$. En el caso de que la propiedad fuera cierta en s_{z-1} verificándose el antecedente *A3*, $s_{z-1}.status.id_i = unknown$ de acuerdo a la propiedad B.2.31. Los efectos de la

acción en este estado no modifican ni st_alg_i ni t_unk_i. Además, no se retiran ni generan mensajes *INF* y en la ruta $ewait(i, last(p\prime).id, p\prime, s_{z-1})$ no se producen cambios. Esto permite concluir que la propiedad sigue cumpliéndose en s_z ya que el consecuente *C3-1* o *C3-2* será cierto por hipótesis inductiva.

- $\pi_z \in \{EndDelArc_j(x)\colon x \in \mathcal{N}\}$. La propiedad sigue verificándose en s_z si el antecedente era falso en s_{z-1} porque no se generan mensajes *ALG* hacia el nodo i, no se modifican las variables asociadas al nodo i ni tampoco el contenido de $set\_waiters_j$. Considerando que en s_{z-1} antecedente y consecuente de la propiedad son ciertos, la ejecución de la acción podría hacer el consecuente falso. Si el antecedente *A1* o *A2* se verifican siendo $s_{z-1}.status.id_j = known$, las variables involucradas tanto en los consecuentes *C1-1* y *C1-3* como en *C2-1* y en *C2-3* no se ven modificadas y los consecuentes siguen siendo ciertos en s_z. En cambio, si se verifican en s_{z-1} el consecuente *C1-2* y su correspondiente antecedente o *C2-2* y el antecedente *A2*, esto es, $s_{z-1}.status.alg_j = dummy$ y $(i, s_{z-1}.ta_i) \in s_{z-1}.setPredToInf_j$, debe comprobarse que, tras la ejecución de la acción, alguno de los consecuentes en los que $s_z.status.id_j = unknown$ se cumple.

  Para el antecedente *A1* podría ser cierto *C1-4* o *C1-5* y para el antecedente *A2*, podría cumplirse en s_z el consecuente *C2-4a* o *C2-4b*. Del nodo j se sabe que: $(i, s_{z-1}.ta_i) \in s_{z-1}.setPred_j$ (propiedad B.2.11), $s_{z-1}.st\_alg_j \neq NULL$ (propiedad B.2.30), $s_{z-1}.t\_unk_j = s_{z-1}.ta_j$ (propiedad B.2.29), $s_{z-1}.status.id_j = known$ (propiedad B.2.15) y por la condición de habilitación $s_{z-1}.blocker_j = x$. Teniendo en cuenta toda esta información, se deduce que el nodo x verifica esta misma propiedad en s_{z-1} con el antecedente *A2* cierto y, por hipótesis inductiva, se debe cumplir el consecuente *C2-3* ($s_{z-1}.status.id_x = known$) o el consecuente *C2-5* ($s_{z-1}.status.id_x = unknown$). El resto de consecuentes se descartan debido a la condición de habilitación de la acción que exige que $(j, s_{z-1}.ta_j, released) \in s_{z-1}.set\_waiters_x$ o bien $s_{z-1}.status.alg_x = aborted$. Combinando la propiedad B.1.5 y la propiedad B.3.2 para la primera opción se confirma que $(j, s_{z-1}.ta_j) \notin s_{z-1}.setPred_x$ y se obtiene la misma conclusión aplicando la propiedad B.2.10 en la segunda opción de habilitación (imposibles consecuentes *C2-4a* y *C2-4b*). Además, usando este resultado y la propiedad B.2.4, se establece que $(j, s_{z-1}.ta_j) \notin s_{z-1}.setPredToInf_x$ siempre que $s_{z-1}.status.id_x$ sea *known* (incompatible con *C2-1* y *C2-2*).

  Cuando se ejecuta la acción y en s_{z-1} existía un mensaje *INF* del nodo x al nodo j (consecuente *C2-3* y $s_{z-1}.status.id_x = known$), este mensaje se mantiene en el canal quedando descrito en los consecuentes *C1-4* o *C2-4b* correspondientes al antecedente *A1* y *A2* respectivamente. Como $s_{z-1}.status.alg_j = dummy$, la propiedad B.2.34 indica que $s_{z-1}.inf\_need_j$ es *false*. Este valor no cambia tras la ejecución de la acción y determina la relación que guarda $sid\prime$ con el campo *sid*

tanto del antecedente *A1* como del antecedente *A2*. En ambos casos el campo *sid* coincide con el que contiene $s_{z-1}.st\_alg_j$ ya que es el que el nodo j, al pasar a *dummy*, transmitió en el mensaje ALG $(A1)$ o quedó almacenado en $s_{z-1}.st\_alg_i$ a la recepción de ese mensaje ALG $(A2)$. Siendo así, resulta obvio que $sid\prime > s_{z-1}.st\_alg_j.sid$ *(C2-3)* y se cumpla que $sid\prime > sid$ tras ejecutarse la acción tal y como señalan los consecuentes *C1-4* o *C2-4b* según el antecedente que se considere.

Si $s_{z-1}.status.id_x = unknown$, el consecuente *C2-5* describe la existencia de un mensaje *INF* con destino el nodo $last(p\prime).id$ siendo $p\prime$ la ruta a partir del cual se define el camino $ewait(j, last(p\prime).id, p\prime, s_{z-1})$. Al ejecutarse la acción, el mensaje *INF* y el camino *ewait* mencionado conservan todas sus características. Por tanto, se confirma que, en s_z, el consecuente *C1-5* asociado al antecedente *A1* es cierto y, del mismo modo, el consecuente *C2-4a* correspondiente al antecedente *A2*.

En caso de que el antecedente *A1*, *A2* o *A3* sea cierto en s_{z-1} y $s_{z-1}.status.id_j = unknown$, la hipótesis inductiva establece que los consecuentes asociados al primer antecedente de la propiedad *C1-4* o *C1-5* son ciertos y, para el segundo antecedente *C2-4a*, *C2-4b* o *C2-5* y para último antecedente *C3-1* o *C3-2*. Como las variables que conforman todos esos consecuentes no se modifican, los mensajes *INF* existentes en el canal no se retiran y no hay cambios en los caminos *ewait*, la propiedad sigue verificándose en s_z.

- $\pi_z = Abort_i$. Los efectos de la acción no retiran del canal el mensaje, pero incrementan el tiempo de activación del nodo i de manera que $m.ta < s_z.ta_i$. Esto implica que la propiedad se verifica con el antecedente falso.

- $\pi_z = Abort_j$. Entre los posibles efectos de esta acción se encuentra la formación de un mensaje *INF*, $m\prime$, dirigido al nodo i y $m\prime.ta = t$, siempre y cuando el par (i, t) pertenezca a $s_{z-1}.setPredToInf_j$. Para que se ejecute la acción es necesario que $s_{z-1}.status.alg_j$ sea *victim* y, según la propiedad B.2.15, entonces $s_{z-1}.status.id_j$ debe ser *known*. Como $m.sid$ coincide con los campos de $s_{z-1}.sim\_id_j$ salvo la última componente que es $s_{z-1}.sim\_id_j.nu.cont_j$, se confirma que este mensaje *INF* cumple que $s_{z-1}.sim\_id_j < m\prime.sid$. Para demostrar que el mensaje *INF* hace cierto alguno de los consecuentes de la propiedad hay que comprobar que el valor de $m\prime.ta$ y de $m\prime.sid$ son adecuados. Si el antecedente *A1* era cierto en s_{z-1} también lo era el consecuente *C1-1* porque la propiedad B.2.6 descarta la existencia de un mensaje *INF* y, por tanto, que el consecuente *C1-3* sea posible. Asumiendo que $t = s_{z-1}.ta_i$ y $m.sid \leq s_{z-1}.sim\_id_j$, se concluye que $m.sid < m\prime.sid$ y $m\prime.ta = s_z.ta_i$. De forma similar se procede si el antecedente cierto en s_{z-1} era *A2*. Considerando el único consecuente posible, *C2-1*, resulta que $t =$

$s_{z-1}.ta_i = s_{z-1}.t\_unk_i$ y $s_{z-1}.st\_alg_i.sid \leq s_{z-1}.sim\_id_j$. Por todo ello, el mensaje *INF* creado cumple que $s_z.st\_alg_i.sid < m\prime.sid$ y $m\prime.ta = s_z.t\_unk_i$. Por último, si el antecedente *A3* se cumple en s_{z-1} se verifica el consecuente *C3-1* y se excluye el *C3-2* por la propiedad B.2.6. Tal y como indica el consecuente *C3-1* debe existir un camino $ewait(i, last(p\prime), p\prime, s_{z-1})$. Suponiendo que el nodo j formara parte de la ruta $p\prime$, $s_{z-1}.status.id_j$ debería ser *unknown* y esto impediría la ejecución de esta acción.

- $\pi_z \in \{Abort_x: x \in \mathcal{N} \setminus \{i, j\}\}$. También es preciso analizar cómo afecta el cambio en la variable $setPredToInf_j$ en los caminos de tipo *ewait* que pudieran existir en s_{z-1}. En el caso de que el nodo j esté contenido en $ewait(i, last(p\prime), p\prime, s_{z-1})$ la definición de *ewait* no es compatible con la condición de habilitación de la acción ya que $s_{z-1}.status.id_j$ debe ser *known* y al formar parte de ese *ewait* debería ser *unknown*. Si se considera el camino $ewait(j, last(p\prime), p\prime, s_{z-1})$ por definición $s_{z-1}.t\_unk_j = t$. Por otra parte, al ejecutarse la acción, $s_{z-1}.status.alg_j$ es *victim* y las propiedades B.2.12 y B.2.26 establecen que $s_{z-1}.st\_alg_j = NULL$ y $s_{z-1}.t\_unk_j = -1$ respectivamente. Como el par (j, t) tiene que estar en el conjunto *setPredToInf* de otro nodo y la propiedad B.2.1 asegura que $t \geq 1$, se deduce que $s_{z-1}.t\_unk_j \neq t$, en contradicción con que el nodo j petenecía a ese *ewait*.

- $\pi_z = initiate_i$. La precondición de la acción señala que $s_{z-1}.status.alg_i$ es *blocked* y $s_{z-1}.status.id_i = known$. La propiedad B.2.13 asegura que no hay ningún elemento en st_alg_i antes de ejecutarse la acción. Además, como los efectos de la acción no cambian esta variable, se puede considerar que tan sólo es posible que el antecedente *A1* sea cierto en s_{z-1}. La hipótesis inductiva concluye en este caso porque las características del antecedente *A1* y de cualquiera de los consecuentes que lleve asociado se conservan intactas.

- $\pi_z = initiate_j$. Al ejecutarse esta acción se puede generar un mensaje ALG, m, con destino el nodo i y $m.ta = s_{z-1}.ta_i$. Este mensaje hace cierto el antecedente *A1* de la propiedad y el consecuente *C1-1* ya que, por efectos de la acción, $m.sid = s_z.sim\_id_j$ y el par $(i, s_z.ta_i)$ se incorpora en $s_z.setPredToInf_j$.

  Dado que la acción modifica la variable *setPredToInf*, se va a comprobar cómo afecta ese efecto en los caminos *ewait* existentes en s_{z-1}. Si es el primer nodo de la ruta se incumple la definición de *ewait* porque el valor de *t_unk* asociado a ese nodo es -1. En caso de que el nodo se sitúe en una posición interior de la ruta, la definición de *ewait* exige que el *status.id* asociado sea *unknown* y la habilitación requiere que sea *known*. En resumen, cuando esta acción está habilitada para un nodo, no existe ningún camino *ewait* que lo contenga.

- $\pi_z = rcvALG_i(j, m)$. Tanto al retirar el mensaje ALG presente en el canal en s_{z-1}, m, como al producirse el efecto que convierte $s_z.status.alg_i$ en *victim* y vacía $s_z.st\_alg_i$, el antecedente de la propiedad podría llegar a ser falso.

  Por otra parte, después de ejecutar la acción puede que $s_z.st\_alg_i \neq \emptyset \wedge s_z.t\_unk_i = s_z.ta_x$. El antecedente *A2* de la propiedad podría ser cierto si $s_z.blocker_i = j$. El mensaje que se almacena procede del mensaje ALG que ocupaba el canal en s_{z-1}. Este mensaje cumplía la propiedad en s_{z-1} por hipótesis inductiva con el antecedente y el consecuente ciertos, porque en el efecto analizado $m.ta = s_{z-1}.ta_i$. Tras la ejecución de la acción, el consecuente de la propiedad sigue siendo cierto ya que $s_z.st\_alg_i.sid = m.sid$. Así mismo, no cambian las variables asociadas al nodo j ni se eliminan mensajes de tipo *INF*. Si existe un camino que cumple $ewait(j, last(p\prime), p\prime, s_{z-1})$, cuando $(i, s_{z-1}.ta_i, released) \in s_{z-1}.set\_waiters_j \wedge s_{z-1}.status.id_j = unknown$, al almacenarse el mensaje ALG habrá un camino que verifica $ewait(i, last(p\prime), p\prime, s_z)$, haciendo cierto el consecuente *C2-4* o *C2-5*.

- $\pi_z \in \{rcvALG_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{ALG}\}$. Un posible efecto de esta acción, cuando $(i, s_{z-1}.ta_i) \in s_{z-1}.setPred_j$ y $s_{z-1}.status.alg_j = blocked$, genera un nuevo mensaje ALG, $m\prime$, dirigido al nodo i tal que $m\prime.ta = s_{z-1}.ta_i$. Este mensaje hace cierto el antecedente *A1* de la propiedad. La creación del mensaje ALG va acompañada de los efectos: $s_{z-1}.status.alg_j = dummy$ y $(i, s_z.ta_i) \in s_z.setPredToInf_j$. Por tanto, el consecuente *C1-2* también pasa a ser cierto.

- $\pi_z \in \{rcvAVS_i(x, m)\colon x \in \mathcal{N} \wedge m \in M_{AVS}\}$. El antecedente *A2* y *A3* podrían llegar a ser falsos cuando los efectos de la acción consisten en el vaciado de $s_z.st\_alg_i$ al pasar $s_z.status.alg_i$ a *victim*.

- $\pi_z \in \{dltINF_i(j, m)\colon m \in M_{INF}\}$. La condición de habilitación exige la presencia de un mensaje *INF* en el canal. Por eso, se van a analizar los casos en los que los consecuentes son ciertos en s_{z-1}. El mensaje *INF* que va del nodo j al nodo i del consecuente *C1-3* verifica $m_{INF}.ta = s_{z-1}.ta_i$. Para que se retire el mensaje por efecto de la acción, $m_{INF}.ta$ debería ser distinto de $s_{z-1}.ta_i$. Así que, la acción no está habilitada. Similarmente, el mensaje *INF* que se describe en el consecuente *C2-3* y en el consecuente *C3-2* cumple que $m_{INF}.ta = s_{z-1}.tunk_i$. En el primer caso, $s_{z-1}.tunk_i = s_{z-1}.ta_i$ y la acción no se puede ejecutar. Aunque en el consecuente *C2-3* $m_{INF}.ta \neq s_{z-1}.ta_i$, la propiedad B.2.2 asegura que $s_{z-1}.t\_unk_i \geq 1$, inhabilitando la acción.

- $\pi_z \in \{dltINF_j(x, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Del mismo modo, los mensajes *INF* dirigidos al nodo j correspondientes a los consecuentes *C1-4* y *C2-4b* cumplen que $m_{INF}.ta = s_{z-1}.t\_unk_i$. En estos casos la acción pdría estar habilitada si

$s_{z-1}.t\_unk_i = -1$, pero aplicando la propiedad B.2.2, resulta que $1 \leq s_{z-1}.t\_unk_i$ se comprueba que finalmente la acción no está habilitada.

- $\pi_z \in \{dltINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Cuando los mensajes *INF* se dirigen al último elemento de un camino que verifica la definición de *ewait*, esto es, $x = last(p\prime)$, se requiere que $m_{INF}.ta = s_{z-1}.t\_unk_{last(p\prime).id}$ (consecuentes *C1-5*, *C2-4a*, *C2-5* y *C3-1* ciertos en s_{z-1}). Como la propiedad B.2.2 establece que $1 \leq m_{INF}.ta$, se tiene que $1 \leq s_{z-1}.t\_unk_{last(p\prime).id}$. Todo ello implica que la acción no se podrá ejecutar en tanto que $s_{z-1}.t\_unk_{last(p\prime).id}$ toma un valor distinto a -1.

- $\pi_z \in \{rcvINF_i(j,\ m)\colon m \in M_{INF}\}$. Aunque el efecto inicial de esta acción consiste en retirar el mensaje *INF* existente en el canal en s_{z-1} y esto podría hacer falso algún consecuente de la propiedad, otro efecto de la acción vacía st_alg_i. Si $s_z.st\_alg_i = NULL$, los antecedentes *A2* y *A3* son falsos verificándose la propiedad. Sin embargo, es posible que en s_{z-1} haya un mensaje *ALG* con destino el nodo i y el tiempo correcto como indica el antecedente *A1*. La hipótesis inductiva establece que en s_{z-1} se verifica el consecuente *C1-3* debido a la existencia del mensaje *INF*, m, en el canal, siendo $m.ta = s_{z-1}.ta_i$. Por otra parte, la condición de habilitación señala que $m.ta = s_{z-1}.t\_unk_i$. Considerando estos dos requisitos resulta que $s_{z-1}.ta_i = s_{z-1}.t\_unk_i$. La propiedad B.2.31 confirma que es imposible que coincidan esos tiempos cuando $s_{z-1}.status.id_i = unknown$ (habilitación) y por tanto, existe una contradicción al suponer que existe un mensaje *AlG* que hace el antecedente *A1* cierto en s_{z-1}.

  Si en s_{z-1} el consecuente *C2-3* o *C3-2* fueran ciertos cumpliéndose, por tanto, la propiedad y se ejecutara la acción, el mensaje *INF* se retiraría del canal, pero tanto el antecedente *A2* y *A3* serían falsos y la propiedad seguiría cumpliéndose en s_z.

- $\pi_z \in \{rcvINF_j(x,\ m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Al ejecutarse esta acción, se puede formar un mensaje *ALG*, $m\prime$, dirigido al nodo i y con $m\prime.ta = s_z.ta_i$. Como la creación del mensaje conlleva que $(i,\ s_z.ta_i) \in s_z.setPredToInf_j$ y $m\prime.sid = s_z.sim\_id_j$, la propiedad se verifica siendo cierto el antecedente *A1* y el consecuente *C1-1*.

  También es posible que al ejecutar esta acción, se genere un mensaje *INF*, $m\prime$, con destino el nodo i y $m\prime.ta = s_{z-1}.ta_i$. Para ello es necesario que $(i,\ s_{z-1}.ta_i) \in s_{z-1}.setPredToInf_j \setminus s_{z-1}.setPred_j$. Además, este nuevo mensaje *INF* cumple que $m\prime.sid > s_z.sim\_id_j$ según la relación de orden establecida. Dado que tras la ejecución de la acción $s_z.status.id_j$ se convierte en *known*, se debería analizar si este mensaje hace que el consecuente *C1-3* o el *C2-3* sea cierto.

En el primer caso, se asume la existencia de un mensaje ALG del nodo j hacia el nodo i que verifica el antecedente *A1*. Dicho mensaje estaba ya presente en el canal en s_{z-1} porque la acción no lo genera. Como (i, $s_{z-1}.ta_i$) $\notin$ $s_{z-1}.setPred_j$, aplicando la propiedad B.2.38, se tiene que (i, $s_{z-1}.ta_i$, *released*) $\in$ $s_{z-1}.set\_waiters_j$. Esta propiedad también considera la posibilidad de que $s_{z-1}.status_i \neq aborted$, pero en las condiciones establecidas en el antecedente, $m.ta = s_{z-1}.ta_i$, la propiedad B.2.8 niega la existencia de un mensaje ALG en el canal dirigido al nodo i. Dado que el tiempo de activación del nodo i no cambia al ejecutar la acción y el mensaje INF del canal en s_{z-1} verifica el consecuente *C1-4*, la propiedad queda demostrada al cumplirse en s_z el consecuente *C1-3*. Si el valor de $s_{z-1}.inf\_need_j$ es *false* entonces $m.sid > sid$. Ejecutando la acción, $s_z.sim\_id_j$ toma el valor de $m.sid$ y $m\prime.sid > s_z.sim\_id_j$, resultando $m\prime.sid > sid$. Cuando $s_{z-1}.inf\_need_j$ tiene el valor *true*, $s_{z-1}.sim\_id_j \geq sid$. La variable $s_z.inf\_need_j$ puede adoptar el valor de $m.sid$ o conservar su valor previo. En cualquiera de estas dos situaciones se obtiene que $m\prime.sid > sid$ ya que $m\prime.sid > s_z.sim\_id_j$.

Si se analiza el segundo caso, se supone que el antecedente *A2* es cierto. La propiedad B.2.39 asegura que, en esas condiciones, (i, $s_{z-1}.ta_i$, *released*) $\in$ $s_{z-1}.set\_waiters_j$. Esto implica que $s_{z-1}.ta_i$ equivale a $s_z.t\_unk_i$. Por otra parte, el mensaje INF del canal, m, se corresponde en s_{z-1} con el descrito en el consecuente *C2-4b*. Dependiendo del valor de $s_{z-1}.inf\_need_j$ o bien $m.sid > sid$ ($s_{z-1}.inf\_need_j = false$) o bien $s_{z-1}.sim\_id_j \geq sid$ ($s_{z-1}.inf\_need_j = true$). Considerando que el efecto de la acción en el caso de *false* hace que $s_z.sim\_id_j$ adopte el valor $m.sid$ y que $m\prime.sid > s_z.sim\_id_j$, resulta $m\prime.sid > sid$. Cuando $s_{z-1}.inf\_need_j = true$, $s_z.sim\_id_j$ puede pasar a valer $m.sid$ o mantener el valor de $s_{z-1}.sim\_id_j$. En ambas posibilidades, también $m\prime.sid > sid$ ya que resulta $m\prime.sid > s_z.sim\_id_j$. Todo esto confirma que el antecedente *A2* y el consecuente *C2-3* son ciertos en s_z.

En otra posible ejecución de la acción la formación de un nuevo mensaje INF no está incluida. Suponiendo que en s_{z-1} la propiedad se verifica con el antecedente *A1* y el consecuente *C1-4*, los efectos de la acción, siendo (i, $s_{z-1}.ta_i$, *received*) $\in$ $s_{z-1}.set\_waiters_j$, permiten que sim_id_x tome un nuevo valor en s_z. Si $s_{z-1}.inf\_need_j$ vale *true*,$s_z.sim\_id_x$ puede conservarse o adquirir el valor de $m.sid$. Dado que en s_{z-1} se cumple el consecuente *C1-4*, se sabe que $s_{z-1}.sim\_id_j \geq sid$. En la primera opción es obvio que $s_z.sim\_id_j \geq sid$ y en la segunda, se comprueba fácilmente que $m.sid > s_{z-1}.sim\_id_j$. Por consiguiente, $s_z.sim\_id_j > s_{z-1}.sim\_id_j$ y $s_z.sim\_id_j > sid$, tal y como requiere la propiedad en el consecuente *C1-1*. En el caso de que $s_{z-1}.inf\_need_j$ sea *false*, $s_z.sim\_id_j = m.sid$. En s_{z-1}, el consecuente *C1-4* establece que $m.sid > sid$. Sustituyendo directamente

en la expresión anterior, resulta que $s_z.sim\_id_j > sid$. Considerando todo ello y observando que $(i, s_z.ta_i) \in s_z.setPredToInf_j$ y $s_z.status.id_j$ es *known*, se confirma que, tras la ejecución, la propiedad también se cumple con el antecedente *A1* y el consecuente *C1-1* ciertos.

- $\pi_z \in \{rcvINF_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Los efectos de esta acción deben ser analizados para todos los casos en los que los consecuentes en s_{z-1} sean ciertos e incluyan mensajes *INF* dirigidos al nodo final de una ruta $p\prime$ ($x = last(p\prime).id$). Esta ruta conforma un camino definido como $ewait(j, last(p\prime).id, p\prime, s_{z-1})$ en los consecuentes *C1-5* y *C2-4a* o $ewait(i, last(p\prime).id, p\prime, s_{z-1})$ en los consecuentes *C2-5* y *C3-1*. A continuación, se comprueba si la propiedad sigue verificándose en s_z cuando los consecuentes: *C1-5*, *C2-4a*, *C2-5* o *C3-1* se cumplen, por hipótesis inductiva, en s_{z-1}.

  En todos estos casos, al ejecutarse la acción, se retira el mensaje *INF* que alcanza el nodo final de la ruta $p\prime$. Entre los efectos de la acción destaca la generación de un mensaje *INF* porque la definición del camino *ewait* existente en s_{z-1} que todos los nodos de la ruta $p\prime$ tienen un par que pertenece a su conjunto *setPredToInf*, pero no a su conjunto *setPred* y ésa es precisamente la condición para la formación del mensaje *INF*. Una vez que se ha creado este mensaje dirigido al nodo de ese par, éste se elimina de *setPredToInf* consiguiendo la reducción de la ruta $p\prime$. Esto implica que el destino del nuevo mensaje *INF* realmente es el nodo que ocupaba la penúltima posición de la ruta $p\prime$ en s_{z-1} y que, al reducirse la ruta, pasa a ser el último nodo de la misma. El resto de variables y requisitos del consecuente en cuestión no se modifican. Por esta razón, el mismo consecuente sigue cumpliéndose en s_z habiéndose reducido la ruta $p\prime$.

  Tras ejecuciones sucesivas de esta acción, en las que el consecuente y el antecedente asociado son ciertos, el mensaje *INF* podría ir dirigido dirigido al primer nodo de la ruta $p\prime$, significando esto que el camino *ewait* se ha eliminado. Las distintas situaciones que se engloban en la ejecución de esta acción tienen en común que el nuevo mensaje *INF* pasa a verificar otro consecuente diferente. Si se parte del consecuente *C1-5*, cuando se extingue el camino *ewait*, se verificará el consecuente *C1-4*. En el caso del consecuente *C2-4a*, se alcanzará el consecuente *C2-4b* y para el consecuente *C2-5*, finalmente se cumplirá el consecuente *C2-3*. En lo que respecta al consecuente *C3-1*, las sucesivas ejecuciones de la acción hacen que el último mensaje *INF* creado, siguiendo los nodos de la ruta $p\prime$, cumpla el consecuente *C3-2*.

  Para demostrar que es correcto el paso del consecuente *C1-5*, *C2-4a*, *C2-5* y *C3-1* al consecuente final *C1-4*, *C2-4b*, *C2-3* y *C3-2* respectivamente, se analizan los cambios que se producen en las variables que intervienen.

A la recepción del mensaje *INF* por el nodo $x = last(p\prime).id$ con la ruta $p\prime$ reducida a dos elementos, se forma un nuevo mensaje *INF* dirigido al nodo j. Además se sabe que, por efectos de la acción, $sid\prime > s_z.sim\_id_{last(p\prime).id}$.

En primer lugar se suponer que $last(p).id \notin nodes(p\prime)$, esto es, el nodo candidato que originó el mensaje *ALG* que viaja hacia el nodo i no se corresponde ni con el nodo j ni con el nodo $last(p\prime).id$. La hipótesis inductiva permite asegurar que, antes de ejecutarse la acción, $m_{INF}.sid > sid$ y la variable *inf_need* asociada a los dos únicos nodos de $p\prime$ tiene como valor *false*. En esta situación, un efecto de la acción consiste en adoptar como valor de $s_z.sim\_id_{last(p\prime).id}$ el valor $m_{INF}.sid$. Combinando adecuadamente esta información, resulta que $sid\prime > sid$. Para concluir que el consecuente *C1-4* se cumple en s_z hay que añadir a lo mencionado que inf_need_j se mantiene como *false* tras ejecutar la acción.

Supóngase ahora que $last(p).id \in nodes(p\prime)$, es decir, el candidato que envió el mensaje *ALG* que va al nodo i está contenido en $p\prime$. Asumiendo que $last(p).id = last(p\prime).id$, se cumple que $s_{z-1}.sim\_id_j \geq sid$ y que a todos los nodos de p les corresponde el valor *false* en la variable *inf_need*, excepto al último, $s_{z-1}.inf\_need_{last(p\prime).id} = true$. Los efectos de la acción en estas circunstancias hacen que o bien $s_z.sim\_id_{last(p\prime).id}$ conserve el valor que tenía en s_{z-1} o bien adopta el valor de $m_{INF}.sid$ porque $m_{INF}.sid > s_{z-1}.sim\_id_{last(p\prime).id}$. En cualquiera de estas posibilidades, se llega a la conclusión que $sid\prime > sid$. El otro requisito que se tiene que comprobar para confirmar que el consecuente *C1-4* se verifica está relacionada con el valor de $s_z.inf\_need_j$. Como esta variable no se modifica en la ejecución de la acción y $s_{z-1}.inf\_need_j = false$ porque el nodo j también está contenido en la ruta p, el consecuente *C1-4* es cierto en s_z. En el caso de que $last(p).id = j$, el consecuente *C1-5* asegura que $s_{z-1}.inf\_need_j = true$ y $s_{z-1}.sim\_id_j \geq sid$. Los efectos de la acción no cambian ninguna de estas variables asociadas al nodo j lo que hace que el consecuente *C1-4* también se cumpla tras la ejecución de la acción.

Esta explicación es válida si se considera que en s_{z-1} es cierto el consecuente *C2-4a* y el campo *sid* que aparece corresponde al mensaje *ALG* almacenado en st_alg_i. La ejecución de la acción, cuando se ha reducido la ruta $p\prime$ a dos elementos, lleva consigo que la propiedad se cumpla en s_z con el consecuente *C2-4b*. El análisis para los consecuentes *C2-5* y *C3-1* es similar salvo por el hecho de que el camino *ewait* comienza en el nodo i y, por tanto, cuando la ruta $p\prime$ contiene sólo dos elementos, el mensaje *INF* es recibido por el nodo $x = j$ y el mensaje *INF* que se forma va dirigido al nodo i. La ejecución de la acción en estos dos casos hace que la propiedad se cumpla con el consecuente *C2-3* y *C3-2*, respectivamente, ciertos.

■

Esta propiedad permite validar la parte de las rutas de mensajes ALG y de mensajes almacenados en *st_alg* que son correctas tras un cambio en el grafo de esperas del sistema. En la propiedad queda constancia de que los mecanismos dinámicos diseñados, para corregir las posibles incongruencias en la información recogida, se ponen en funcionamiento.

Propiedad B.2.55. Sea $\{j, k\} \subseteq \mathcal{N}$, sea $sid \in T$, sea $p \in P$ tal que $p = (n_1, t_1)(n_2, t_2) \ldots (n_m, t_m)$. Si $\exists\ i$ con $1 < i \leq m$ tal que $(n_{i-1}, t_{i-1}) \in s.setPred_{n_i}$ entonces

- $((ALG, s.ta_j, sid, p) \in s.channel(k, j) \Rightarrow \forall\ l < i$: $(n_l, s.ta_{n_l}) \in s.setPred_{n_{l+1}} \wedge s.status.alg_{n_l} = dummy \wedge (j, s.ta_j) \in s.setPred_{n_1} \wedge s.state_j = blocked \wedge n_1 = k) \wedge$

- $(s.st\_alg_j = (sid, p) \Rightarrow \forall\ l < i$: $(n_l, s.ta_{n_l}) \in s.setPred_{n_{l+1}} \wedge s.status.alg_{n_l} = dummy \wedge ((s.status.alg_j = dummy \wedge n_1 = j) \vee ((j, s.ta_j) \in s.setPred_{n_1} \wedge s.status.alg_j = candidate)))$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.setPred_i = \emptyset$ por lo que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_x(y)$: $\{x, y\} \subseteq \mathcal{N}\}$. Al ejecutarse la acción $s_z.status.alg_x \in \{candidate, blocked\}$. Este cambio de estado del nodo x podría hacer falso el consecuente de la propiedad. Sin embargo, como la acción se habilita siendo $s_{z-1}.state_x = active$ o, por la propiedad B.3.3, $s_{z-1}.status.alg_x = active$, la hipótesis inductiva establece que la acción sólo está habilitada para el nodo n_i o para nodos localizados posteriormente a éste en las rutas consideradas en la propiedad. Esto es así porque se sabe que el estado de los nodos que aparecen en posiciones previas a n_i es *dummy* en s_{z-1}. En consecuencia, tras la ejecución de la acción y, a pesar del cambio de $status.alg_x$, el consecuente sigue siendo cierto en s_z porque no se modifican las variables citadas en la propiedad: $setPred_x$, st_alg_x y ta_x y, tampoco se retiran mensajes de tipo ALG.

  Si el nodo x coincide con el nodo j que incluye la propiedad y, además el consecuente de la propiedad es cierto en s_{z-1}, es posible que, en s_{z-1}, exista un mensaje ALG dirigido al nodo j o haya almacenada una ruta válida en $s_{z-1}.st\_alg_j$. En cualquiera de los dos casos, la acción no estará habilitada porque es necesario que $s_{z-1}.state_j = active$ (por la propiedad B.3.3, $s_{z-1}.status.alg_j = active$). En el primer caso, la propiedad señala que $s_{z-1}.state_j$ debe ser *blocked* y en el segundo, que $s_{z-1}.status.alg_j$ debería ser *dummy* o *candidate*.

  También es posible que, por efecto de la acción, se genere un mensaje ALG. La ruta de dicho mensaje cuenta con un único elemento y, por tanto, no cumple los

requisitos establecidos en el antecedente de la propiedad. El resto de variables no se modifican, así que la propiedad se cumple por hipótesis inductiva en esta situación.

- $\pi_z \in \{EndAddArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. El efecto fundamental de esta acción consiste en añadir un nuevo elemento, $(y,\ t)$ al conjunto $setPred_x$, pero también es posible que se genere un mensaje ALG. Así que, es necesario comprobar que ninguno de estos efectos afectan al cumplimiento de la propiedad. El resto de las variables que aparecen en la propiedad no se modifican al ejecutar la acción.

  El mensaje que puede crearse cuando $s_{z-1}.status.alg_x = dummy$ adopta como ruta la del mensaje almacenado en $s_{z-1}.st\_alg_x$. Como la propiedad se cumplía en s_{z-1} por hipótesis inductiva y la propiedad B.2.30 establece que el primer nodo de la ruta almacenada en $s_{z-1}.st\_alg_x$ es precisamente el nodo x, para asegurar que el mensaje ALG creado cumple la propiedad, basta con observar que $(y,\ t) \in s_z.setPred_x$. Considerando la propiedad B.3.2, que indica que en esas circunstancias $(y,\ t,\ received) \in s_z.set\_waiters_x$, y la propiedad B.1.4, se llega a la conclusión de que $t = s_z.ta_y$ y $s_z.state_y = blocked$. Por tanto, la propiedad es cierta en s_z. Si se genera un mensaje ALG siendo $s_{z-1}.status.alg_x = candidate$, su ruta cuenta con un único elemento, $(x,\ s_z.ta_x)$. Como no existe ningún elemento anterior a ése, es evidente que el antecedente de la propiedad es falso.

  Cuando el nodo x se corresponde con el nodo j y existe en s_{z-1} un mensaje ALG dirigido a éste, el mensaje se conserva en s_z con las mismas características verificando también la propiedad. Por otro lado, si st_alg_j en s_{z-1} contiene una ruta que hace que el consecuente sea cierto, al ejecutar la acción, no se producen cambios en las variables que intervienen en el consecuente y la propiedad se verifica.

  La ejecución de la acción también podría hacer cierto el antecedente de la propiedad para cualquier ruta que ya incluyera el par $(y,\ t)$, dado que éste se incorpora a $s_z.setPred_x$. Las únicas acciones que, en un estado anterior, pudieron añadir este elemento a cualquiera de las rutas de interés son las acciones: $StartAddArc_y(n)$ o $EndAddArc_y(n,\ t_n)$, $initiate_y$, $rcvALG_y(n,\ m\prime)$ o $rcvINF_y(n,\ m\prime)$.

  El mensaje ALG generado en estas acciones, a excepción de la acción $rcvALG_y(n,\ m)$, se envía a un nodo que en ese momento pertenece al conjunto $setPred_y$. La ruta de ese mensaje ALG cumple la propiedad con el antecedente falso porque sólo consta de un elemento. Por otra parte, como ninguna de las acciones consideradas modifican el tiempo de activación del nodo y, se asume que el tiempo asociado al nodo y en la ruta es el actual. Cuando el mensaje ALG llega a su

destino, de acuerdo con su estado, puede ser almacenado en el correspondiente *st_alg* y enviado un nuevo mensaje ALG. Para que se pudiera ejecutar la acción $EndAddArc_x(y,\ t)$ en s_z, $(y,\ t,\ sent) \in s_{z-1}.set\_waiters_x$. Esto implica que se ha tenido que ejecutars la acción $StartAddArc_y(x)$ y, a su vez, el requisito de esta acción obliga a que el estado del nodo y sea *active*. En consecuencia, el nodo y debería desbloquearse cambiando su tiempo de activación. En resumen, la acción $EndAddArc_x(y,\ t)$ no se puede ejecutar siendo t el mismo tiempo con el que aparece en una ruta existente.

Podría suceder que antes de que el mensaje ALG llagara a su destino, se hubiera eliminado la espera de este nodo por el nodo y. En esta situación concreta el consecuente y el antecedente de la propiedad serían falsos ya que el mensaje al retirarse del canal no puede formar ninguna nueva ruta.

En lo que respecta a la ejecución de la acción $rcvALG_y(n,\ m)$ siendo el estado del nodo y *blocked* se asume que anteriormente ese mensaje ALG fue generado por cualquiera de las acciones ya mencionadas. Estas acciones sólo envían mensajes ALG a nodos que pertenecen al conjunto de predecesores de nodo n. Por lo tanto, $(y,\ t)$ tenía que pertenecer en ese instante a $setPred_n$. De acuerdo con la propiedad B.3.2, $(y,\ t,\ received)$ pertecería a $set\_waiters_n$. La propiedad B.1.4 establece que entonces $blocker_y = n$ y $t\_activ_y = t)$ o, por el contrario, $state_y = aborted$.

- $\pi_z \in \{StartDelArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Los efectos de esta acción eliminan el par $(y,\ t)$ del conjunto $setPred_x$, pero no cambian el valor de ta_x, $status.alg_x$ ni de st_alg_x y no crean ni eliminan mensajes ALG. Por consiguiente, si la propiedad se supone cierta en s_{z-1} con el antecedente falso, la ejecución de la acción mantiene a falso el antecedente en s_z. Suponiendo que es el consecuente cierto en s_{z-1}, se sabe que en ningún nodo de las rutas que se indican previo a n_i la acción puede estar habilitada porque su estado es *dummy*. Según el medio, la acción se ejecuta siempre que $(y,\ t,\ received) \in s_{z-1}.set\_waiters_x$ y bien $s_{z-1}.state_x$ sea *active* o bien $s_{z-1}.state_y$ sea *aborted*. Para otros nodos de la ruta, la acción se podría ejecutar, pero los efectos no afectarían a las variables asociadas a los nodos que aparecen en el consecuente de la propiedad. De igual modo, considerando la otra posibilidad de habilitación, también se puede deducir que la acción sólo podría estar habilitada cuando el nodo y se corresponda con el nodo n_i de las rutas de la propiedad o ocupe en ellas posiciones superiores a n_i. Incluso en estos casos de habilitación, los efectos de la acción no modifican las características de los nodos que hacen el consecuente cierto en s_{z-1} y se concluye que la propiedad se verifica en s_z. También es posible que, al ejecutarse la acción, $s_z.st\_alg_x = NULL$. Este efecto podría convertir el antecedente en falso cumpliéndose así la propiedad.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x, y\} \subseteq \mathcal{N}\}$. Si la propiedad se cumplía en s_{z-1} con el antecedente falso, como los efectos de la acción no introducen ningún cambio que lo pueda hacer cierto: no se generan nuevos mensajes ALG, no se almacena una nueva ruta en st_alg_x, $setPred_x$ no varía y el tiempo de activación del nodo x se incrementa impidiendo conservar su valor en s_{z-1}, la hipótesis inductiva concluye. Además, al ejecutarse la acción $s_z.st\_alg_x$ podría ser vaciado, el antecedente de la propiedad también podría llegar a ser falso.

  Por otro parte, la condición de habilitación de esta acción indica que $(x, t, released) \in s_{z-1}.set\_waiters_y$. Aplicando la propiedad B.1.5 se puede asegurar que entonces, $(x, t, received) \notin s_{z-1}.set\_waiters_y$ o, según la propiedad B.3.2, $(x, t) \notin s_{z-1}.setPred_y$. En caso de que la propiedad se cumpliera en s_{z-1} con el consecuente cierto, la acción estaría habilitada únicamente para aquellos nodos que estuvieran situados en la rutas consideradas en la propiedad en la posición del nodo n_i o posteriormente. Al ejecutarse la acción en esos casos, se incrementa el tiempo de activación del nodo x y el estado del nodo x pasa a ser *active*, pero esto no influye en las características exigidas a los nodos que aparecen en el consecuente de la propiedad. Así que, el consecuente de la propiedad sigue siendo cierto en s_z.

  Finalmente, cuando el nodo x coincide con el nodo j siendo el antecedente y el consecuente ciertos en s_{z-1} se plantean varias situaciones que hay que analizar. Suponiendo que existiera en s_{z-1} un mensaje AlG que hubiera partido del nodo y y fuese dirigido al nodo j y a su vez el nodo j perteneciera a $s_{z-1}.setPred_y$, la habilitación de la acción sería imposible. En caso de que el mensaje tuviera como origen un nodo distinto al nodo y, la ejecución de la acción incrementaría el tiempo de activación del nodo j haciendo el antecedente de la propiedad falso ya que el tiempo incluido en el mensaje ALG dejaría de ser correcto. En lo que respecta a una ruta almacenada en $s_{z-1}.st\_alg_j$, se tendría que verificar que en s_{z-1} el par $(j, s_{z-1}.ta_j)$ estuviera incluido en el conjunto $setPred$ del siguiente nodo en la ruta si el ocupa la primera posición o del primer nodo de la ruta si no está incluido. En ambas situaciones, la habilitación de la acción es imposible porque el nodo j no puede pertenecer al conjunto de predecesores del nodo y ni de ningún otro nodo hasta que se active y eso modifica su tiempo de activación.

- $\pi_z \in \{Abort_x : x \in \mathcal{N}\}$. Los efectos de esta acción consisten básicamente en el vaciado del conjunto $setPred_x$, el incremento del tiempo de activación del nodo x y el cambio de estado del nodo x a *aborted*. A continuación se estudia la influencia de estos efectos en el cumplimiento de la propiedad. La precondición de la acción indica que $s_{z-1}.status.alg_x = victim$. Esto conlleva que la acción sólo puede estar habilitada para nodos que ocupen en las rutas consideradas la posición del nodo n_i y posteriores.

En el caso de que la propiedad se verifique en s_{z-1} con el antecedente falso, la propiedad seguirá cumpliéndose en s_z con el antecedente falso. Esto se debe a que la ejecución de la acción no incluye ningún par en el conjunto $setPred_x$ que hiciera cierto el antecedente siendo falso. Nótese que si $x = n_i$ es obvio que el antecedente de la propiedad es falso porque $s_z.setPred_{n_i} = \emptyset$. Por otra parte, si la propiedad se cumple en s_{z-1} siendo cierto el consecuente, se concluye aplicando la hipótesis inductiva. Como los efectos de la acción para los nodos en los que está habilitada no afectan a las variables asociadas a los nodos que se citan en el consecuente de la propiedad, el consecuente sigue siendo cierto.

Dado que la acción está habilitada cuando $s_{z-1}.status.alg_x = victim$ y la propiedad B.2.12 establece que $s_{z-1}.st\_alg_x = NULL$, sólo hay que tener en cuenta la posibilidad de que en s_{z-1} exista un mensaje ALG dirigido al nodo x y que cumpla la propiedad. Analizando los efectos de la acción cuando el nodo x coincide con el nodo j y existe un mensaje ALG dirigido al nodo j cumpliendo, se llega a la conclusión de que el consecuente pasa a ser falso porque $s_z.status.alg_j = aborted$ y la propiedad B.3.5 indica que entonces $s_z.state_j = aborted$. Sin embargo, el antecedente también se convierte en falso debido a que, tras la ejecución de la acción, el tiempo de activación del nodo j se incrementa dejándose de cumplir la condición temporal del mensaje ALG. De acuerdo con esta explicación, la propiedad se verifica en s_z.

- $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$. Al ejecutarse la acción sólo se modifica $status.alg_x$ que pasa a ser *candidate* siendo $s_{z-1}.status.alg_x = blocked$, pero en ese estado es imposible que la acción estuviera habilitada para alguno de los nodos de interés de la propiedad. Por todo ello, la propiedad es cierta en s_z.

  Un posible efecto de esta acción consiste en la generación de mensajes ALG. Como la ruta de dichos mensajes cuenta con un único elemento, el antecedente de la propiedad es falso para este caso.

  Si el nodo x se corresponde con el nodo j que aparece en la propiedad y la acción está habilitada, se debe estudiar la posibilidad de que existiera en s_{z-1} un mensaje ALG dirigido al nodo j o hubiera una ruta almacenada en $s_{z-1}.st\_alg_j$ que verificaran la propiedad. La ejecución de la acción no afecta al mensaje y el cambio de estado del nodo j no influye porque la propiedad B.3.4 asegura que, cuando $s_z.status.alg_j = candidate$, $s_z.state_j = blocked$. Según esto, la propiedad se cumple en s_z. El otro supuesto resulta ser imposible porque la habilitación de la acción requiere que $s_{z-1}.status.alg_j = blocked$ y $s_{z-1}.status.id_j = known$. De acuerdo con esas condiciones, la propiedad B.2.13 asegura que $s_{z-1}.st\_alg_j = NULL$ y, por tanto, no existe ninguna ruta almacenada en $s_{z-1}.st\_alg_j$ que pueda analizarse.

- $\pi_z \in \{rcvALG_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{ALG}\}$. El primer efecto de esta acción consiste en retirar del canal el mensaje m, que tenía por destino el nodo x en s_{z-1}. Al eliminarse el mensaje del canal, la propiedad sigue cumpliéndose, pero con el antecedente falso.

  El efecto de la acción que consiste en almacenar en $s_z.st\_alg_x$ la ruta íntegra del mensaje ALG recibido tiene lugar cuando $s_{z-1}.status_x = candidate$. Aplicando la hipótesis inductiva, se concluye que la propiedad se verifica en s_z porque no se modifican ninguna de las variables asociadas al nodo x que se incluyen en la propiedad. Si el nodo x se corresponde con el nodo j, se observa que el primer nodo de la ruta tiene que coincidir con el origen del mensaje ALG, esto es, $n_1 = y$. Dado que la variable $setPred_{n_1}$ no se modifica por efectos de esta acción, $s_{z-1}.ta_j = s_z.ta_j$ y $s_z.status.alg_j = candidate$, la propiedad se cumple en esta situación.

  Otro efecto similar al anterior se produce cuando $s_{z-1}.status.alg_x = blocked$. En este caso la ruta que se almacena en $s_z.st\_alg_x$ es la del mensaje ALG, cumple la propiedad en s_{z-1}, antecedida por el par $(x,\ s_z.ta_x)$. Como resulta que $s_z.status.alg_x$ pasa a ser $dummy$, el consecuente de la propiedad se hace cierto en s_z. Cuando el nodo x coincide con el nodo j, al ejecutarse la acción, la propiedad se verifica de igual modo porque $s_z.status.alg_j = dummy$.

  Si la ruta del mensaje ALG cumple la propiedad en s_{z-1} debido a que hace falso el antecedente porque toda la ruta apuntada es falsa y los nodos que aparecen en ella se han activado y cambiado su tiempo de activación, al añadir un elemento en la primera posición de esa ruta puede ocurrir que: el antecedente y el consecuente se verifiquen para el primer elemento de la ruta ($(x,\ s_z.ta_x) \in s_{z-1}.setPred_y$ y $s_z.status_x = dummy$) o el antecedente siga siendo falso porque $(x,\ s_z.ta_x) \notin s_{z-1}.setPred_y$.

  Otro de los posibles efectos de la acción es la generación de un mensaje ALG, $m\prime$. La ruta de este nuevo mensaje coincide con la ruta que se guarda en $s_z.st\_alg_x$. Como ya se ha comprobado que $s_z.st\_alg_x$ cumple la propiedad, se concluye que este nuevo mensaje ALG también. Si el nodo x coincide con el nodo j citado en la propiedad, se observa que el primer nodo de la ruta de $m\prime$ es precisamente el nodo j. Por otra parte, se cumple que el estado del nodo j pasa a ser $dummy$, el par $(j,\ s_z.ta_j)$ sigue perteneciendo al conjunto $setPred$ del nodo que aparecía en la primera posición de la ruta del mensaje ALG retirado del canal (nodo y). Además el destino de este nuevo mensaje es un nodo id que pertenece a $s_z.setPred_j$ y para confirmar que la variable $s_z.state_{id} = blocked$ se recurre a propiedades del medio. Considerando la propiedad B.3.2, se puede afirmar que $(id,\ t,\ received) \in s_z.set\_waiters_j$. La propiedad del medio B.1.4 establece entonces que $s_z.state_{id} \in \{blocked,\ aborted\}$. Si se cumple que $s_z.t\_activ_{id} = t$ o,

por la propiedad B.3.1, $s_z.ta_{id} = t$, $s_z.state_{id} = blocked$ y $m\prime.ta = s_z.ta_{id}$. Por tanto, la propiedad se cumple en s_z con el antecedente y consecuente ciertos. En caso de que $s_z.state_{id} = aborted$ el consecuente de la propiedad será falso, pero a su vez, B.1.4 señala que $s_z.t\_activ_{id} > t$. En consecuencia, $m\prime.ta \neq s_z.ta_{id}$ y el antecedente de la propiedad también será falso. Esto implica que la propiedad se verifica en cualquier caso para el nuevo mensaje *ALG*.

Por último, al ejecutarse la acción, puede producirse un cambio de estado que haga que $s_z.status.alg_x = victim$. Dado que este estado sólo se puede alcanzar si $s_{z-1}.status.alg_x = candidate$, la acción estará habilitada cuando el nodo x se corresponda con el nodo n_i o con nodos que aparecen en posiciones posteriores a éste para las rutas válidas según la propiedad. En estas circunstancias, los efectos no modifican las variables involucradas en la propiedad y, por tanto, la propiedad seguirá siendo cierto en s_z ya que, por la hipótesis inductiva, se asume que la propiedad es cierta en s_{z-1}. Analizando este mismo efecto, cuando el nodo x se corresponde con el nodo j, se evidencia que el antecedente de la propiedad podría llegar a ser falso porque el cambio de estado a *victim* va acompañado por otro efecto tal que $s_z.st\_alg_j = NULL$.

- $\pi_z \in \{rcvINF_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Uno de los posibles efectos de la acción es la generación de un mensaje *ALG*. Como la ruta de este mensaje está formada por un único elemento, este mensaje verifica la propiedad con el antecedente falso. Otro de los efectos de la acción que debe ser analizado es el cambio de estado por el que $s_z.status.alg_x = candidate$. Sabiendo que ese cambio se produce cuando $s_{z-1}.status.alg_x = blocked$ y $s_{z-1}.setPredToInf \neq \emptyset$, se deduce que sólo si el nodo x ocupa posiciones posteriores al nodo n_i mencionado en la propiedad será posible el efecto. Esto conlleva que el cambio de estado no afecta a ninguno de los nodos anteriores a n_i y, por hipótesis inductiva, la propiedad se cumple para ellos.

  Analizando los efectos de la acción cuando el nodo x se corresponde con el nodo j, resulta obvio que el antecedente de la propiedad podría llegar a ser falso porque hace que $s_z.st\_alg_j = NULL$. En el caso de que existiese un mensaje *ALG* que verificase la propiedad en s_{z-1} la acción no afecta al cumplimiento de la propiedad, ya que al ejecutarse no lo retira del canal ni modifica la ruta. En cuanto al estado del nodo j, la condición de habilitación de la acción indica que $s_{z-1}.status.id_j = unknown$. La propiedad B.2.15 establece que $s_{z-1}.status.alg_j \in \{active, blocked\}$. La hipótesis inductiva obliga a que $s_{z-1}.state_j = blocked$. Dado que las propiedades B.3.3 y B.3.4 señalan que sólo es posible que $s_{z-1}.status.alg_j = blocked$ y los efectos de la acción pueden hacer que $s_z.status.alg_j \in \{blocked, candidate\}$, es evidente que $s_z.state_j = blocked$ (propiedad B.3.4). Por tanto, la propiedad se cumple en s_z en este supuesto.

- $\pi_z \in \{rcvAVS_x\ (y,\ m)$: $\{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Aunque al ejecutarse esta acción puede cambiar el estado del nodo x, considerando los nodos que aparecen en la ruta en posiciones previas al nodo n_i, se concluye que es imposible que este efecto se produzca porque la hipótesis inductiva indica que $s_{z-1}.status.alg_x = dummy$ y el efecto requiere que $s_{z-1}.status.alg_x = candidate$.

  Si el nodo x coincide con el nodo j de la propiedad y $s_{z-1}.status.alg_j = candidate$ y la propiedad se cumplía en s_{z-1}, se observa que tras la ejecución de la acción la propiedad sigue verificándose en s_z. Como la acción no genera ni retira del canal mensajes ALG, la estructura de cualquier mensaje ALG que se dirigiera al nodo j no se modifica y tampoco cambia el conjunto $setPred_j$ ni ta_j, tan sólo es necesario estudiar el efecto que hace que $s_z.status.alg_j = victim$. Este cambio de estado no influye en el cumplimiento de la propiedad porque la propiedad B.3.4 asegura que entonces $s_z.state_j = blocked$. Además, este efecto va acompañado de otro que vacía st_alg_j y que no afecta porque eso podría hacer el antecedente la propiedad falso.

■

Si el borrado de esperas provoca que la ruta de un mensaje ALG o de un mensaje almacenado en st_alg pase a ser parcialmente válida, habrá un nodo a partir del cual toda la inforamción sea incorrecta. Esta propiedad resume todos los estados posibles en los que este nodo puede hallarse y describe la relación que guarda su identidad simulada con la de otros nodos registrados en las rutas consideradas.

Propiedad B.2.56. Sea $\{j,\ k,\ x\} \subseteq \mathcal{N}$, sea $\{sid,\ sid\prime \in T$ y sea $\{p,\ p\prime\} \in P$ tal que $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_m,\ t_m)$.

Si $(ALG,\ s.ta_j,\ sid,\ p) \in s.channel(k,\ j) \vee s.st\_alg_j = (sid,\ p) \wedge \exists\ i$ con $1 < i \leq m$ tal que $(n_{i-1},\ t_{i-1}) \in s.setPred_{n_i} \wedge (n_i,\ t_i) \notin s.setPred_{n_{i+1}} \Rightarrow$

- $((s.status.alg_{n_i} \in \{candidate,\ victim,\ active\} \wedge s.status.id_{n_i} = known \wedge s.sim\_id_{n_i} \geq sid) \vee$
- $(s.status\_id_{n_i} = unknown \wedge s.inf\_need_{n_i} = true \wedge s.sim\_id_{n_i} \geq sid \wedge$
  - $((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \vee$
  - $(ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(x,\ last(p\prime).id))))$

  $\vee$
- $((s.status.alg_{n_i} = dummy \vee (s.status.id_{n_i} = unknown \wedge s.inf\_need_{n_i} = false)) \wedge$

- $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
- $(ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel(x,\ last(p\prime).id)) \wedge$
  - $((last(p).id \in nodes(p\prime) \wedge s.sim\_id_{last(p).id} \geq sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p) - last(p))$: $s.inf\_need_x = false \wedge s.inf\_need_{last(p).id} = true)$ $\vee$
  - $(last(p).id \notin nodes(p\prime) \wedge sid\prime > sid \wedge \forall\ (x,\ t_x) \in visited\_nodes(p\prime)$: $s.inf\_need_x = false))))))$

Demostración: En el estado inicial, s_0, $\forall\ \{i,\ j\} \subseteq \mathcal{N}$ se cumple que $s_0.st\_alg_j = NULL$ y $s_0.channel(i,\ j) = \epsilon$ por lo que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_x(y)$: $\{x,\ y\} \subseteq \mathcal{N}\}$. Entre los efectos de la acción está la creación de un mensaje ALG. Como su ruta contiene sólo un elemento, la propiedad se verifica con el antecedente falso.

  Si la acción está habilitada en s_{z-1} siendo $s_{z-1}.status.id_{n_i} = known$, la propiedad B.2.47 establece que, cuando $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$, $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPredToInf_{n_i}$. Esto implica que $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$ y, por tanto, sólo se puede producir el efecto que se deriva de esta situación. La condición de habilitación de la acción supone que $s_{z-1}.state_{n_i} = active$ o, según la propiedad B.3.3, $s_{z-1}.status.alg_{n_i} = active$. Si $s_{z-1}.status.id_{n_i} = known$, los efectos de la acción cambian a *candidate* la variable $status.alg_{n_i}$, pero no modifican el valor de $s_{z-1}.sim\_id_{n_i}$. En consecuencia, el consecuente de la propiedad sigue siendo cierto aplicando la hipótesis inductiva. En el caso de que $s_{z-1}.status.id_{n_i}$ sea *unknown*, la hipótesis inductiva asegura que en el canal hay un mensaje INF dirigido al nodo n_i o al último nodo de un camino *ewait*, $last(p\prime).id$. Las demás características que se cumplen en s_{z-1} dependen del valor $s_{z-1}.inf\_need_{n_i}$ y la ejecución de la acción no varía ninguna de ellas, quedando demostrada la propiedad en s_z.

  Para nodos que aparezcan en una ruta en posiciones anteriores al nodo n_i la acción no se puede ejecutar porque su estado es *dummy* de acuerdo con la propiedad B.2.55. Por otra parte, en los nodos que se corresponden a un camino $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$ se puede ejecutar la acción, pero los efectos de la misma no afectan al cumplimiento de la propiedad en los nodos que ésta hace referencia.

- $\pi_z \in \{EndAddArc_x(y,\ t)$: $\{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Esta acción está habilitada según el medio cuando $(y,\ t,\ sent) \in s_{z-1}.set\_waiters_x$ o, por la propiedad B.1.5, $(y,\ t,\ received) \notin s_{z-1}.set\_waiters_x$.

De los efectos de la acción que afectan a las variables de la propiedad, destaca la incoproración del par $(y,\ t)$ en $s_z.setPred_x$ ya que podría hacer que alguna ruta existente en s_{z-1} llegara a cumplir el antecedente de la propiedad. Para que esto fuera posible el nodo n_i de la propiedad en s_{z-1} debería ser el nodo y y así, tras la ejecución de la acción, el nodo n_i se correspondería, con el nodo x.

Para que el par $(y,\ t)$ aparezca en un ruta de las indicadas en la propiedad junto con al menos otro elemento es necesario que, en un estado previo al de la habilitacón de la acción, el nodo y haya recibido un mensaje ALG. A su vez para que se produzca este hecho es imprescindible que el elemento $(y,\ t)$ estuviera en $setPred_x$ o lo que es lo mismo, por la propiedad B.3.2, $(y,\ t,\ received)$ fuera un elemento de $set\_waiters_x$. Aplicando la propiedad B.2.1 y la propiedad B.3.1, se tiene que t puede ser menor o igual que ta_y o t_activ_y, respectivamente. Así que, si la acción debe estar habilitada posteriormente, es preciso que se suceda la ejecución de las siguientes acciones: $StartDelArc_x(y,\ t)$, $EndDelArc_y(x)$ y finalmente $StartAddArc_y(x)$. Uno de los efectos de la acción $EndDelArc_y(x)$ incrementa el tiempo de activación del nodo y, lo que hace imposible que, en el momento de ejecutarse la acción que se analiza, el nodo y tenga asociado el mismo tiempo que el registrado supuestamente en la ruta. De este modo queda demostrado que esta situación no se puede producir.

La acción además puede generar mensajes ALG siempre que $s_{z-1}.status.alg_x \in \{dummy,\ candidate\}$. En el primer estado posible la ruta del nuevo mensaje ALG es una copia de la ruta contenida en $s_{z-1}.st\_alg_x$. La hipótesis inductiva permite concluir que la propiedad también se cumple en s_z para la ruta del ALG. Por otro lado, siendo *candidate* el nodo x, la ruta del mensaje ALG que se forma contiene un único elemento. Es obvio que entonces se cumple la propiedad con el antecedente falso.

El análisis de la acción se completa, suponiendo que el nodo x pertenece a una ruta de las que indica la propiedad y aparece en posiciones previas al nodo n_i. Como el conjunto $s_z.setPred_x$ se ve aumentado, pero no se elimina el par que hacía cierta la propiedad en s_{z-1} ni modifica las variables correspondiente al nodo n_i, la propiedad sigue verificándose. Del mismo modo, considerando que el nodo x estuviera localizado en la ruta de un camino *ewait* en el que el nodo n_i ocupa la primera posición, la acción podría ejecutarse. El único efecto de la acción que se produce en este supuesto también consiste en añadir $(y,\ t)$ a $s_z.setPred_x$ porque los nodos de esta ruta, según la definición de *ewait*, cumplen que $s_{z-1}.status.id_x = unknown$. A pesar de esto, los nodos implicados en la propiedad siguen verificando los requisitos exigidos en s_{z-1}.

- $\pi_z \in \{StartDelArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Una de las posibles habilitaciones de la la acción consiste, según el medio, en que $(y,\ t,\ received) \in s_{z-1}.set\_waiters_x$

y $s_{z-1}.state_y = aborted$. Aplicando las propiedades B.3.2 y B.3.5 respectivamente se obtiene la condición equivalente: $(y, t) \in s_{z-1}.setPred_x$ y $s_{z-1}.status.alg_y = aborted$. De acuerdo a esta condición de habilitación, la acción no se podrá ejecutar si el nodo x coincide con el nodo n_i o se posiciona antes que él en las rutas porque la propiedad B.2.55 establece que en esos casos el estado del nodo y, que precede en las rutas al nodo x, es *dummy*. Así mismo, si el nodo x ocupa posiciones más allá del nodo n_i en la ruta $p\prime$ de un camino *ewait*, no será posible ejecutar la acción porque, $(y, t) \notin s_{z-1}.setPred_x$ por definición de camino *ewait* y la habilitación de la accón requiere lo contrario.

Analizando la otra condición de habilitación, $(y, t) \in s_{z-1}.setPred_x$ y $s_{z-1}.status.alg_x = active$, se estudian por separado los efectos de la acción según el valor de $s_{z-1}.status.id_x$.

Cuando $s_{z-1}.status.id_x = known$, es obvio que sólo interesa comprobar el cumplimiento de la propiedad para el caso de que el nodo x sea precisamente n_i. Para posiciones del nodo x en la ruta anteriores al nodo n_i, la acción no está habilitada porque el estado del nodo x sería *dummy* (propiedad B.2.55). De igual manera, si se supone que el nodo x forma parte del camino $ewait(n_i, last(p\prime).id, p\prime, s_{z-1})$, no se cumple la condición porque los nodos de la ruta $p\prime$, según definición deben ser *unknown*.

Si $x = n_i$, el par (y, t) deja de pertenecer al conjunto $setPred_{n_i}$ al ejecutarse la acción. En consecuencia, el nodo n_i al que hace referencia la propiedad pasa a ser el nodo y. Como el estado del nodo y no cambia por los efectos de la acción, mantiene el que le correspondía en s_{z-1}, esto es, de acuerdo con la propiedad B.2.55 $s_z.status.alg_y = dummy$. Así mismo, la propiedad B.2.47 establece que el par (y, t) estaba también incluido en $s_{z-1}.setPredToInf_{n_i}$. Por este motivo, se genera un mensaje *INF* como efecto de la acción que va del nodo n_i hacia el nodo y. Para verificar la propiedad el nodo y debe cumplir que el valor de $sid\prime$ del nuevo mensaje *INF* sea superior al del campo *sid* de las rutas p consideradas. Como $sid\prime$ se construye a partir de $s_{z-1}.sim\_id_x$, se sabe que $sid\prime > s_{z-1}.sim\_id_x$. Considerando la hipótesis inductiva, se asume además que $s_{z-1}.sim\_id_x > sid$. Por consiguiente, resulta que $sid\prime > sid$ y la propiedad queda demostrada.

En el caso de que $s_{z-1}.status.id_x = unknown$, la acción sólo puede estar habilitada si el nodo x se corresponde con n_i. Los efectos de la acción en ese estado no generan ni eliminan mensajes de tipo *ALG* o *INF* y tampoco almacenan o eliminan mensajes de ningún *st_alg*. Tan sólo cabe mencionar que si en s_{z-1} existiera un mensaje *INF* asociado a la existencia de un camino $ewait(n_i, last(p\prime).\text{id}, p\prime, s_{z-1})$, tras la ejecución de la acción, el mensaje *INF* permanecería en el canal, pero el camino *ewait* se ampliaría. El nuevo camino *ewait* se iniciaría en el nodo y debido al único efecto que se produce al ejecutar la acción consistente en

retirar el par $(y\ ,\ t)$ de $setPred_{n_i}$. Para comprobar que la propiedad se verifica en s_z basta con analizar que se cumplen las características de un camino *ewait* encabezado por el nodo y: $(y\ ,\ t) \notin setPred_{n_i}$, dado que $(y,\ t) \in s_{z-1}.setPred_{n_i}$, la propiedad B.2.47 confirma que $(y,\ t) \in s_{z-1}.setPredToInf_{n_i}$, $s_z.status.id_{n_i} = unknown$ y $s_{z-1}.t\_unk_y = t$ ya que el nodo y es *dummy* en s_{z-1} (propiedad B.2.55), de acuerdo con la propiedad B.2.29 $s_{z-1}.t\_unk_y = s_{z-1}.ta_y$ y de nuevo por la propiedad B.2.55 $s_{z-1}.ta_y = t$. Como resulta que la ejecución de la acción que se analiza no cambia los valores de $setPredToInf_{n_i}$, $status.id_{n_i}$, t_unk_y, el camino $ewait(y,\ last(p\prime).id,\ p\prime,\ s_z)$ cumple la definición.

Hay un efecto que cambia el valor de $sim\_id_{n_i}$ a $(n_i,\ s_z.ta_{n_i},\ \epsilon)$. Se puede apreciar que esto sucede cuando $s_z.setPred_{n_i} = \emptyset$ y, por tanto, el nodo n_i no puede aparecer en ninguna ruta existente en s_z (antecedente falso). La modificación de esta variable no se puede producir para ningún otro nodo x que ocupe una posición distinta a la del nodo n_i porque no cumple las condiciones que lo hacen posible.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Una de las precondiciones del medio indica que $(x,\ t,\ confirmed) \in s_{z-1}.set\_waiters_y$ o, por la propiedad B.1.5, $(x,\ t,\ received) \notin s_{z-1}.set\_waiters_y$. Aplicando la propiedad B.3.2 resulta entonces que $(x,\ t) \notin s_{z-1}.setPred_y$.

  Suponiendo que el nodo x coincide con el nodo n_i, se analizan diversos casos dependiendo del valor de $s_{z-1}.status.alg_{n_i}$. Si $s_{z-1}.status.alg_{n_i} = dummy$, el antecedente de la propiedad establece que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$ y, por la propiedad B.2.47, $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPredToInf_{n_i}$, es decir, $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$. De acuerdo con esto, el efecto de la acción convierte $s_z.status.id_{n_i} = unknown$. Al ser el nodo n_i *dummy* en s_{z-1} la propiedad B.2.34 establece que $s_{z-1}.inf\_need_{n_i} = false$. Dado que, después de ejecutarse la acción, el valor de esta variable no varía, la propiedad sigue verificando cualquiera de los dos posibles consecuentes que se cumplían en s_{z-1} para el estado *dummy*.

  Suponiendo que $s_{z-1}.status.alg_{n_i} = candidate$ y $s_{z-1}.inf\_need_{n_i} = false$, los efectos de la acción hacen que $s_z.status.alg_{n_i}$ pase a ser *active*, pero no modifican la variable $sim\_id_{n_i}$ ni otras incluidas en el mismo consecuente que debe verificarse para ambos estados del nodo n_i. Por otro lado, si $s_{z-1}.status.alg_{n_i} = candidate$ y $s_{z-1}.inf\_need_{n_i} = true$, tras la ejecución de la acción, se alcanza el estado $s_z.status.id_{n_i} = unknown$ y se mantiene el valor de $inf\_need_{n_i}$ a *true*. En esta situación debería cumplirse cualquiera de las opciones que establece la propiedad para ese estado concreto.

  Como en los dos consecuentes posibles existe un mensaje *INF* en un canal y la acción no los genera, los mensajes *INF* ya estaban presente en ese canal en

s_{z-1}. La existencia de esos mensajes queda confirmada aplicando la propiedad B.2.54 con el antecedente *A2* cierto. Se deduce que el antecedente *A2* se cumple en s_{z-1} porque: a partir de la condición de habilitación de la acción se sabe que $s_{z-1}.blocker_{n_i} = y$, la propiedad B.2.35 señala que $s_{z-1}.st\_alg_{n_i} \neq \emptyset$ siendo $s_{z-1}.inf\_need_{n_i} = true$ y, finalmente, la propiedad B.2.29 indica que $s_{z-1}.t\_unk_{n_i} = s_{z-1}.ta_{n_i}$. El consecuente asociado al antecedente *A2* de la propiedad B.2.54 depende del valor de $s_{z-1}.status.id_y$. Si $s_{z-1}.status.id_y = known$ en s_{z-1} se verifica el consecuente *C2-3* porque, asumiendo que $(n_i, t_i) \notin s_{z-1}.setPred_y$, la propiedad B.2.4 señala que $(n_i, t_i) \notin s_{z-1}.setPredToInf_y$. El mensaje *INF* que se describe en el consecuente *C2-3* de la propiedad B.2.54 procede del nodo y y se dirige al nodo n_i, con característica temporal $s_{z-1}.t\_unk_x$ y $sid\prime > s_{z-1}.st\_alg_{n_i}.sid$. En el caso de que $s_{z-1}.status.id_y = unknown$, el consecuente *C2-5* de la propiedad B.2.54 considera la existencia de un camino *ewait* y de un mensaje *INF* con destino $last(p\prime)$.id que se ajusta a los requisitos que exige la propiedad que se analiza.

Si se comprueba la propiedad cuando $s_{z-1}.status.alg_{n_i} = blocked$ y $s_{z-1}.status.id_{n_i} = unknown$, es obvio que, la propiedad sigue cumpliéndose para cualquier valor de $s_{z-1}.inf\_need_{n_i}$. Los efectos de la acción no retiran ni modifican los mensajes *INF* y no producen cambios en ninguna ruta de las que considera la propiedad. Además, los cambios en variables asociadas al nodo n_i no afectan al cumplimiento de la propiedad, incluido el cambio de estado del nodo n_i a *active*.

A continuación, se analiza la posibilidad de que la acción esté habilitada para un nodo x que esté localizado en una ruta $p\prime$ de un camino *ewait* y no sea el nodo n_i de la propiedad. En este supuesto, según la definición de *ewait*, $s_{z-1}.status.id_x = unknown$ y $s_{z-1}.t\_unk_x = t$. Por otro lado, la propiedad B.2.15 establece en estas condiciones que $s_{z-1}.status.alg_x \in \{active, blocked\}$. Cuando $s_{z-1}.status.alg_x = active$ la acción no tiene efectos, pero cuando $s_{z-1}.status.alg_x = blocked$, la propiedad se cumple en s_{z-1} de maneras distintas dependiendo del valor de $s_{z-1}.inf\_need_x$. A pesar de verificarse distintos consecuentes en s_{z-1}, se concluye que cada uno se sigue cumpliendo en s_z dado que los efectos de la acción mantienen los mensajes *INF* existentes y las características de cualquier ruta de las consideradas en esta propiedad.

También se puede suponer que la acción está habilitada, según el medio, cuando $s_{z-1}.state_y = aborted$. En este caso es evidente que $s_{z-1}.setPred_y = \emptyset$ y, en consecuencia, $(x, t) \notin s_{z-1}.setPred_y$. La propiedad B.2.10 establece que, si el estado del nodo y es *aborted* entonces, $s_{z-1}.setPredToInf_y = \emptyset$. Esto implica que ni el nodo x ni el nodo y pueden formar parte de la ruta $p\prime$ a partir de la cual se definen los caminos *ewait* mencionados en la propiedad. Así mismo, se omite el estudio del caso $x = n_i$ porque $s_{z-1}.setPredToInf_y = \emptyset$. Al ejecutar la acción,

esta variable no cambia y se mantiene como falso el antecedente de la propiedad en s_z.

Por último, se debe mencionar que la acción no está habilitada siempre que el nodo x aparezca en la ruta p en posiciones previas al nodo n_i. De acuerdo con la propiedad B.2.55, el nodo x debería pertenecer al conjunto $setPred_y$ y la habilitación de la acción impone lo opuesto.

- $\pi_z \in \{Abort_x\colon x \in \mathcal{N}\}$. La acción no está habilitada para ningún nodo x que aparezca en la ruta p en posiciones anteriores al nodo n_i. Como la propiedad B.2.55 asegura que $\forall\ l < i,\ s_{z-1}.status.alg_{n_l} = dummy$, la acción no puede ejecutarse porque es necesario que sea *victim*.

  En el caso de que el nodo x se localice en la ruta p en posiciones superiores al nodo n_i, el nodo x podrá formar parte del camino $ewait(n_i, last(p\prime), p\prime, s_{z-1})$. De acuerdo con la definición de este tipo de camino, $s_{z-1}.status.id_x = unknown$. Sin embargo, la precondición de la acción obliga a que $s_{z-1}.status.alg_x = victim$ y la propiedad B.2.15 establece que $s_{z-1}.status.id_x = known$. Por tanto, tampoco la acción está habilitada en este caso.

  Si el nodo x coincide con el nodo n_i y la acción se ejecuta , se sabe que $s_{z-1}.status.alg_{n_i} = victim$, $s_{z-1}.status.id_{n_i} = known$ y $s_{z-1}.inf\_need_{n_i} = false$. La hipótesis inductiva asegura que $s_{z-1}.sim\_id_x \geq sid$.

- $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$. La condición de habilitación de esta acción consiste en que $s_{z-1}.status.alg_x = blocked$ y $s_{z-1}.status.id_x = known$. Por tanto, el nodo n_i no puede ejecutar la acción si ya aparece en una ruta de las indicadas en la propiedad. Si el nodo x no pertenece a ninguna ruta, se puede formar un mensaje ALG. Como la ruta de este nuevo mensaje está constituida por un único elemento, el antecedente de la propiedad es falso. Por otra parte, la acción no está habilitada si el nodo x aparece en en posiciones anteriores al nodo n_i o en posteriores en la ruta de un camino *ewait*. En el primer caso, la propiedad B.2.55 establece que $\forall\ l < i,\ s_{z-1}.status.alg_{n_l} = dummy$ y en la segunda situación, se sabe, por la definición de un camino *ewait* que $s_{z-1}.status.id_x = unknown$. Estos valores hacen imposible la ejecución de la acción.

- $\pi_z \in \{rcvINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Para que quede habilitada esta acción es necesario que $s_{z-1}.status.id_x = unknown$. Teniendo en cuenta la propiedad B.2.55, se deduce que el nodo x puede coincidir con el nodo n_i que se menciona en la propiedad o con cualquier nodo que forme parte de la ruta $p\prime$ y que constituye un camino *ewait*.

  Entre los posibles efectos de la acción se encuentra la creación de un mensaje ALG. Como la ruta de ese mensaje sólo cuenta con un elemento la propiedad se

verifica en este caso con el antecedente falso. Al ejecutarse la acción el mensaje *INF* existente en el canal en s_{z-1} se elimina, pero pueden generarse nuevos mensajes *INF*. Como además se producen cambios en la identidad simulada del nodo x, seguidamente se analizan estos efectos considerando que $x = n_i$ y según el valor de la variable $s_{z-1}.inf\_need_{n_i}$.

Si $s_{z-1}.inf\_need_{n_i} = true$, tras la ejecución de la acción, la identidad simulada del nodo n_i puede cambiar, $s_z.sim\_id_{n_i} = m_{INF}.sid$ o conservar su valor, $s_z.sim\_id_{n_i} = s_{z-1}.sim\_id_{n_i}$. Dado que en s_{z-1} se cumplía la propiedad y, por tanto, $s_{z-1}.sim\_id_{n_i} \geq sid$, se llega a la conclusión de que en ambas situaciones $s_z.sim\_id_{n_i} > sid$. Para comprobarlo en el primer caso, basta con darse cuenta de que $s_z.sim\_id_{n_i} > s_{z-1}.sim\_id_{n_i}$ y en el segundo caso es obvio. Así mismo, hay que analizar el estado que alcanza el nodo n_i al ejecutar la acción. Sabiendo que los efectos de la acción hacen que $s_z.status.alg_{n_i} = known$ y que, por ser el antecedente cierto en s_{z-1}, $s_{z-1}.setPred_{n_i} \neq \emptyset$, se deduce que $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$ aplicando la propiedad B.2.47. A pesar de que no se den las condiciones para formar un nuevo mensaje *INF*, $s_z.setPredToInf_{n_i} \neq \emptyset$ y la propiedad B.2.43 asegura que $s_z.status.alg_{n_i}$ no puede ser *blocked*. Por tanto, los posibles estados finales del nodo n_i son *active* o *candidate*. Como ya ha quedado comprobado que $s_z.sim\_id_{n_i} > sid$, la propiedad se verifica en cualquiera de los estados mencionados.

Cuando $s_{z-1}.inf\_need_{n_i} = false$ y existe un mensaje *INF* en s_{z-1} cumpliendo la propiedad, los efectos de la acción hacen que $s_z.sim\_id_{n_i}$ adopte el valor del campo *sid* que contiene el mensaje *INF* que se recibe. Al igual que en el caso anterior, $m_{INF}.sid > sid$. Así que, tras la ejecución de la acción se evidencia que $s_z.sim\_id_{n_i} > sid$. Por otro lado, los efectos de la acción modifican tanto la variable $status.alg_{n_i}$ como la variable $status.id_x$. Los estados alcanzables son: $s_z.status.alg_{n_i} \in \{active, blocked, candidate\}$ siendo $s_z.status.id_{n_i} = known$. De acuerdo con la propiedad B.2.43, si $s_z.status.alg_{n_i} = blocked$ y $s_z.status.id_{n_i} = known$ obliga a que $s_z.setPredToInf_{n_i} = \emptyset$. Si el antecedente de la propiedad era cierto en s_{z-1}, el par (n_{i-1}, t_{i-1}) pertenecía a $s_{z-1}.setPred_{n_i}$ y, por la propiedad B.2.47, ese par también tenía que estar incluido en $s_{z-1}.setPredToInf_{n_i}$. De todo ello se deriva que es imposible que el nodo n_i alcance el estado *blocked*. En los otros estados, $s_z.status.alg_{n_i} \in \{active, candidate\}$, la propiedad se verifica en s_z para el nodo n_i porque $s_z.sim\_id_{n_i} > sid$ tal y como ya se había comprobado.

Para que se forme un nuevo mensaje *INF* en cualquiera de las dos situaciones descritas es imprescindible que exista un par (id, t) que esté contenido en $s_{z-1}.setPredToInf_{n_i}$ pero no en $s_{z-1}.setPred_{n_i}$. Dado que el antecedente de la propiedad establece que el nodo n_{i-1} pertenece al conjunto $s_{z-1}.setPred_{n_i}$, la

generación de un nuevo mensaje *INF* sólo tendrá lugar si el par $(id,\ t)$ no coincide con el nodo correspondiente n_{i-1} de la ruta p que se cita en la propiedad. Por consiguiente, el mensaje *INF* que se pudiera crear ejecutando la acción no afecta a los nodos involucrados en la propiedad.

Teniendo en cuenta que los mensajes *INF* que van dirigidos al último elemento de la ruta $p\prime$, a partir de la cual se define el camino *ewait* mencionado en la propiedad, resulta obvio que sólo tiene interés estudiar los efectos de la acción cuando el nodo x se corresponde con $last(p\prime).id$. Nuevamente, el análisis va a depender del valor de la variable *inf_need* para el nodo $last(p\prime).id$ en s_{z-1}. Si $s_{z-1}.inf\_need_{last(p\prime).id}$ es *true*, la hipótesis inductiva establece que $s_{z-1}.sim\_id_{n_i} \geq sid$. Cuando se ejecute la acción, se enviará un nuevo mensaje *INF* a un nodo id tal que $(id,\ t) \in s_{z-1}.setPredToInf_{last(p\prime).id} \setminus s_{z-1}.setPred_{last(p\prime).id}$. Al enviar este mensaje, el par $last(p\prime)$ se elimina del camino *ewait* y pasando a ser el nuevo $last(p\prime).id$ el destino del mensaje *INF* generado. A pesar de ello, el consecuente de la propiedad seguirá siendo cierto en s_z porque la identidad simulada del nodo n_i no varía al ejecutarse la acción y, en consecuencia, $s_z.sim\_id_{n_i} \geq sid$. Tras sucesivas ejecuciones de la acción a cargo de un nodo x que forme parte del camino *ewait*, este camino llegará a desaparecer y el último mensaje *INF* emitido se asimilará al analizado anteriormente cuando $x = n_i$ y que cumplía la propiedad.

En el caso en que $s_{z-1}.inf\_need_{last(p\prime).id}$ es *false* hay que estudiar dos situaciones diferentes: $last(p).id \in nodes(p\prime)$ o $last(p).id \notin nodes(p\prime)$. Si $last(p).id \in nodes(p\prime)$, la propiedad establece que $s_{z-1}.sim\_id_{last(p).id} \geq sid$ y para todos los nodos de la ruta p, a excepción de su último nodo, su *inf_need* es *false* siendo $s_{z-1}.inf\_need_{last(p).id} = true$. Al recibirse el mensaje *INF*, se reducirá la ruta $p\prime$ del camino *ewait* y se formará un nuevo mensaje *INF* que verifique lo mismo que el retirado del canal. Esto sucederá de igual forma hasta que $last(p\prime).id = last(p).id$ y la ruta $p\prime$ esté totalmente contenido en la ruta p. Cuando el par $last(p)$ queda eliminado del camino *ewait* debido a la formación del correspondiente mensaje *INF*, $last(p).id \notin nodes(\mathrm{p}\prime)$ y $last(p\prime).id$ será un nodo distinto a $last(p).id$ que pertenece a *ewait* y está contenido en la ruta p.

A partir de ese momento, como $s_{z-1}.inf\_need_{n_i} = false$ y $s_{z-1}.inf\_need_{last(p).id} = true$, se tiene que verificar que $sid\prime > sid$. Por la relación de orden, se sabe que $sid\prime > s_z.sim\_id_{last(p\prime).id}$. Así que, existirán distintas situaciones según sea el valor de la identidad simulada de $last(p\prime).id$: $s_z.sim\_id_{last(p\prime).id} = s_{z-1}.sim\_id_{last(p\prime).id}$ o $s_z.sim\_id_{last(p\prime).id} = m_INF.sid$. Sustituyendo en $sid\prime > s_z.sim\_id_{last(p\prime).id}$ y en la relación de identidades simuladas del estado previo, $s_{z-1}.sim\_id_{last(p).id} \geq sid$, se obtiene que $sid\prime > sid$, como se quería demostrar. Por otra parte, la variable *inf_need* de cualquier nodo que aparece entre los pares de $visited\_nodes(p\prime)$

tiene el valor *false*. En el estado anterior el único nodo con *inf_need* a *true* era precisamente $last(p).id$. Después de retirar este nodo del camino *ewait*, el valor de *inf_need* del resto de nodos que componen $p\prime$ en s_z es *false* como indica la propiedad.

Mientras existan elementos en el camino *ewait*, se generará un nuevo mensaje *INF* como el que se acaba de analizar. Sin embargo, cuando el mensaje *INF* creado se dirija al nodo n_i, la estructura del mensaje *INF* verificará el consecuente de la propiedad en el que $s_z.status.id_{n_i} = unknown$, $s_z.inf\_need_{n_i} = false$ y ya no existe una ruta $p\prime$ sobre la que esté construido un camino *ewait*. Para confirmar el cumplimiento de la propiedad sólo es necesario comprobar que $sid\prime > sid$ y ya quedó demostrado.

Por último, cuando $last(p).id \notin nodes(p\prime)$, el consecuente de la propiedad verifica en s_{z-1} que $m_{INF}.sid > sid$ y la variable *inf_need* de todos los nodos que constituyen la ruta $p\prime$ es *false*. En esta situación la ruta $p\prime$ está contenida totalmente en la ruta p y, de acuerdo a los efectos de la acción, se genera un nuevo mensaje *INF* al nodo correspondiente en el orden establecido en el camino *ewait*. La identidad simulada del mensaje, $sid\prime$, verifica la condición establecida en la propiedad, $sid\prime > sid$. Para comprobar esto, sólo es necesario observar que $sid\prime > s_z.sim\_id_{last(p\prime).id}$ (relación de orden). Si $s_{z-1}.inf\_need_{last(p\prime).id} = false$, entonces $s_z.sim\_id_{last(p\prime).id} = sid$. Sustituyendo convenientemente en las expresiones resulta que el mensaje *INF*, cumple la condición asociada al campo *sid*. Cuando ya no existan elementos en el camino *ewait*, se generará un mensaje *INF* hacia el nodo n_i. Su estructura responderá al consecuente de la propiedad que fue analizado anteriormente.

- $\pi_z \in \{dltINF_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Dado que la precondición de esta acción exige que $m \in s_{z-1}.channel(y, x)$, se analizan los casos en los que el consecuente es cierto debido a la existencia de un mensaje *INF* en el canal. Si $x = n_i$ y $m.ta = s_{z-1}.t\_unk_x$, la acción estará habilitada cuando $s_{z-1}.t\_unk_x = -1$. Aplicando la propiedad B.2.2, se obtiene que $1 \leq m.ta \leq s_{z-1}.ta_x$. Al sustituir $m.ta$ por $s_{z-1}.t\_unk_x$, resulta evidente que $s_{z-1}.t\_unk_x$ no puede valer -1. En consecuencia, la acción que se analiza no está habilitada. En el caso de que el mensaje *INF* vaya dirigido al último elemento de un camino *ewait*, o sea, $x = last(p\prime).id$, se concluye del mismo modo.

- $\pi_z \in \{rcvAVS_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Al ejecutarse esta acción se puede producir un cambio de estado para el nodo x. Cuando el nodo x aparece en una ruta p en posiciones previas al nodo n_i o en una ruta $p\prime$ en posiciones posteriores al nodo n_i, la acción no está habilitada. En el primer caso, la propiedad B.2.55 indica que $\forall\ l < i\ s_{z-1}.status.alg_{n_l} = dummy$ y en el segundo caso, la

definición del camino $ewait(n_i, last(p\prime).id, p\prime, s_{z-1})$ establece que $s_{z-1}.status.id_x = unknown$. Como la precondición de la acción consiste en que $s_{z-1}.st\_avsrps_x \neq NULL$ y, según las propiedades B.2.20 y B.2.17,esto implica que $s_{z-1}.status.alg_x = candidate$ y $s_{z-1}.status.id_x = known$, la acción no se puede ejecutar. Si el nodo x coincide con el nodo n_i y $s_{z-1}.status.alg_x = candidate$, tras la ejecución de la acción, podría alcanzarse un nuevo estado, $s_z.status.alg_x = victim$. La propiedad sigue verificándose porque las variables del consecuente mantienen su valor y, tanto para el estado previo como para el nuevo estado se exigen los mismos requisitos.

- $\pi_z = sndAVS_{n_i}$ o $\pi_z = sndAVSRSP_{n_i}$. A pesar de que los efectos de estas acciones modifican la variable $inf\_need_{n_i}$ y pasa a valer *true*, la hipótesis inductiva permite afirmar que la propiedad se cumple porque ambas acciones están habilitadas cuando $s_{z-1}.st\_avsrps_{n_i} \neq NULL$. Aplicando las propiedades B.2.20 y B.2.17, se concluye que entonces $s_{z-1}.status.alg_{n_i} = candidate$ y la variable $inf\_need_{n_i}$ no influye en este caso. Así mismo, como las variables involucradas no se ven afectadas, la propiedad se cumple.

- $\pi_z \in \{sndAVS_x\colon x \in \mathcal{N}\}$ o $\pi_z \in \{sndAVSRSP_x\colon x \in \mathcal{N}\}$. Cuando el nodo x aparece en una ruta p en posiciones previas al nodo n_i o en una ruta $p\prime$ en posiciones posteriores al nodo n_i, la acción no está habilitada. En el primer caso, la propiedad B.2.55 indica que $\forall\ l < i\ s_{z-1}.status.alg_{n_l} = dummy$ y en el segundo caso, la definición del camino $ewait(n_i, last(p\prime).id, p\prime, s_{z-1})$ establece que $s_{z-1}.status.id_x = unknown$. La precondición de la acción consiste en que $s_{z-1}.st\_avsrps_x \neq NULL$ y esto implica que, según las propiedades B.2.20 y B.2.17,$s_{z-1}.status.alg_x = candidate$ y $s_{z-1}.status.id_x = known$.

■

De acuerdo con esta propiedad, en el sistema no puede haber dos nodos con identidades simuladas iguales. Además, las identidades simuladas cuando se propagan gracias a los mensajes *INF* (campo *sid*), tampoco pueden coincidir con la identidad simulada de ningún nodo, ni con el campo *sid* de otros mensajes *INF* del sistema.

Propiedad B.2.57. Sea $(id, t, \gamma) \in T$, se verifican simultáneamente las aserciones *A1* y *A2*.
A1: $\exists\ i \in \mathcal{N}$ tal que $s.sim\_id_i = (id, t, \gamma) \Rightarrow$

- *C1-1*: $\forall\ j \in \mathcal{N}$ tal que $j \neq i$ se verifica que $s.sim\_id_j \neq (id, t, \gamma) \wedge$
- *C1-2*: $\forall\ \{j, k\} \subseteq \mathcal{N}, \forall\ m \in M_{INF}$, si $m \in s.channel(j, k)$ entonces $m.sid \neq (id, t, \gamma) \wedge$

- *C1-3*: Si $\gamma = \alpha{\cdot}\beta$ tal que $\alpha \in \mathbb{N}^*$ y $\beta \in \mathbb{N}^+ \wedge \exists\ j \in \mathcal{N}$ que verifica $s.sim\_id_j = (id,\ t,\ \alpha)$ entonces $s.cont_j > first(\beta)$

A2: $\exists\ \{i,\ j\} \subseteq \mathcal{N} \wedge \exists\ m \in M_{INF}$ tal que $m \in s.channel(i,\ j)$ verificando $m.sid = (id,\ t,\ \gamma) \Rightarrow \forall\ \beta \in \mathbb{N}^*$:

- *C2-1*: $\forall\ k \in \mathcal{N}$ se verifica que $s.sim\_id_k \neq (id,\ t,\ \gamma{\cdot}\beta) \wedge$
- *C2-2*: $\forall\ \{k,\ l\} \subseteq \mathcal{N}$: $k \neq i \vee l \neq j$, $\forall\ m \in M_{INF}$, si $m \in s.channel(k,\ l)$ entonces $m.sid \neq (id,\ t,\ \gamma{\cdot}\beta) \wedge$
- *C2-3*: Si $\gamma = \alpha{\cdot}\delta$ tal que $\alpha \in \mathbb{N}^*$ y $\delta \in \mathbb{N}^+ \wedge \exists\ k \in \mathcal{N}$ que verifica $s.sim\_id_k = (id,\ t,\ \alpha)$ entonces $s.cont_k > first(\delta)$

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.sim\_id_i = (i,\ t,\ \epsilon)$ y $\forall\ \{j,\ k\} \subseteq \mathcal{N}$, $s_0.channel(j,\ k) = \epsilon$. Como todos los nodos tienen identificadores distintos, el consecuente *C1-1* es cierto. Por otro lado, como no hay ningún mensaje en los canales de comunicación, el antecedente *A2* es falso y el consecuente *C1-2* es cierto. También se puede comprobar que el consecuente *C1-3* es cierto porque ϵ no puede expresarse como $\alpha{\cdot}\beta$ siendo $\beta \neq \epsilon$. A continuación, se analizan los efectos de las acciones que modifican en s_z la variable *sim_id* y generan mensajes *INF*.

- $\pi_z \in \{StartDelArc_x(y,\ t){:}\ \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. En el caso de que el conjunto $setPred_x$ quede vacío tras la ejecución de la acción, $s_z.status.alg_x = active$ y $s_z.status.id_x = known$, la variable $s_z.sim\_id_x$ adopta el valor $(x,\ s_z.ta_x,\ \epsilon)$. Este valor no puede aparecer en s_z en el campo *sid* de ningún mensaje *INF* ni en la variable *sim_id* de ningún otro nodo porque, al estar en una ruta el nodo tuvo que estar bloqueado para enviar o recibir mensajes *ALG* y desde que el nodo x pasó a *active* y cambió su tiempo de activación (efectos de la acción $EndDelArc_x(n)$), el nodo i no ha podido difundir su identidad con su tiempo de activación actual.

  Esta acción también genera mensajes *INF*. El campo *sid* se construye en ellos a partir de $s_{z-1}.sim\_id_x$. Aunque las dos primeras componente de ese campo son similares en todos los mensajes que se envían, la última componente es diferente en todos ellos porque se añade un número distinto como sufijo a $s_{z-1}.sim\_id_x.nu$. Si $s_{z-1}.sim\_id_x = (id,\ t,\ \gamma)$ y $s_{z-1}.cont_x = c$, el campo *sid* de los mensajes generados será $(id,\ t,\ \gamma{\cdot}c)$ para el primer mensaje *INF* incrementándose en una unidad hasta alcanzar $(id,\ t,\ \gamma{\cdot}(c{+}n{-}1))$ para el enésimo mensaje *INF*.

  Suponiendo que este mensaje *INF* viola el consecuente *C1-2*, es decir, en s_z existe un nodo cuyo identidad simulada en su última componente contiene $\gamma{\cdot}(c{+}b)$, como la acción no modifica ese valor se deduce que ese nodo ya existía en s_{z-1}. La hipótesis inductiva asegura que el consecuente *C1-3* también se cumplía en

s_{z-1} para ese nodo. Así que $s_{z-1}.sim\_id_x = (id,\ t,\ \gamma)$ y, en ese mismo estado, hay otro nodo con identidad simulada $(id,\ t,\ \gamma{\cdot}(c{+}b))$. Considerando que ese nodo es precisamente el nodo i y verifica el consecuente *C1-3*, esto es, $\alpha = \gamma$ y $\beta = (c{+}b)$ y además existe un nodo j que coincide con el nodo x cumpliendo que $s_{z-1}.sim\_id_x = (id,\ t,\ \alpha)$, se obtiene que $s_{z-1}.cont_x > (c{+}b)$. Esto contradice el hecho de que $s_{z-1}.cont_x = c$.

Igualmente considérese que no se cumple en s_z el consecuente *C2-1*, es decir, $\exists\ k \in \mathcal{N}$ tal que $s_z.sim\_id_k = (id,\ t,\ \gamma{\cdot}\beta)$. Como la acción no modifica el valor de sim_id_k, se mantiene el valor que ya tenía en s_{z-1}. Para la creación del mensaje *INF* se emplean las variables $s_{z-1}.sim\_id_x = (id,\ t,\ \chi)$ y $s_{z-1}.cont_x = c$. Por tanto, el campo *sid* del primer mensaje *INF* generado por la acción resulta ser $(id,\ t,\ \chi c)$. La condición se violaría si $\gamma = \chi{\cdot}c$. Aplicando el consecuente *C1-3* en s_{z-1} y habiendo asumido la existencia de un nodo, k, cuya identidad simulada en s_{z-1} es $(id,\ t,\ \chi{\cdot}c{\cdot}\beta)$, siendo $\gamma = \chi{\cdot}c$, se concluye que $s_{z-1}.cont_x > c$ lo que resulta ser imposible.

En el caso de que el consecuente *C2-2* no sea cierto en s_{z-1}, el campo *sid* del primer mensaje *INF* generado por la acción resulta ser $(id,\ t,\ \chi{\cdot}c)$, siendo $s_{z-1}.sim\_id_x = (id,\ t,\ \chi)$ y $\gamma = \chi{\cdot}c$. Al mismo tiempo, hay otro mensaje cuyo campo *sid* es $(id,\ t,\ \gamma{\cdot}\beta)$. Esto contradice el cumplimiento del consecuente *C1-3* ya que señala que en s_{z-1} un mensaje *INF* dispone del campo $(id,\ t,\ \chi{\cdot}c{\cdot}\beta)$ por lo que si $s_{z-1}.cont_x = c$ no puede ser superior a $first(c{\cdot}\beta)$.

Finalmente, supóngase que el consecuente *C2-3* no se cumple en s_z, esto es, el campo *sid* del mensaje *INF* es $(id,\ t,\ \gamma)$, pero existe un nodo k cuya identidad simulada es $(id,\ t,\ \alpha)$ con $s_z.cont_k \leq first(\delta)$, siendo $\gamma = \alpha \cdot \delta$ tal que $\alpha \in \mathbb{N}^*$ y $\delta \in \mathbb{N}^+$. Para la formación del mensaje *INF* las variables presentan los siguientes valores: $s_{z-1}.sim\_id_x = (id,\ t,\ \chi)$ y $s_{z-1}.cont_x = c$. Esto implica que la última componente del campo *sid* del mensaje *INF* es $\gamma = \chi{\cdot}c$. Por consiguiente, $\chi{\cdot}c$ coincide con $\alpha{\cdot}\delta$, siendo $\delta \neq \epsilon$. Aplicando la hipótesis inductiva, el consecuente *C1-3* asegura que $s_{z-1}.sim\_id_x = (id,\ t,\ \alpha\beta)$ y $c > first(\beta)$. Por tanto, como $\delta = \beta{\cdot}c$ con $\beta \in \mathbb{N}^*$, $c > first(\delta)$ y resulta imposible que $s_z.cont_k \leq first(\delta)$.

- $\pi_z = Abort_x$. Al igual que en la acción precedente, los mensajes *INF* que pueden mandarse al ejecutarse esta acción poseen un campo *sid* distinto porque su última parte es siempre diferente. A $s_{z-1}.sim\_id_x.nu$ se le añade el contenido de la variable $cont_x$ y este contador se va incrementando en una unidad cada vez que se hace un envío, mandando así $m.sid$ diferentes. Los consecuentes *C1-2*, *C1-3* y los asociados al antecedente *A2* se verifican del mismo modo que en la acción $StartDelArc_x(y,\ t)$ cuando se analizó la creación de mensajes *INF*. Además, tras la ejecución de la acción, el tiempo de activación del nodo x varía y este tiempo es el que se incluye en la nueva identidad simulada, $(x,\ s_z.ta_x,\ \epsilon)$, que

adquiere el nodo x una vez abortado. Estos efectos no afectan al cumplimiento de la propiedad.

- $\pi_z \in \{rcvINF_x(y, m)\colon x \in \mathcal{N} \wedge m \in M_{INF}\}$. Esta acción permite, en algunos casos, cambiar de identidad simulada al nodo x. En esas situaciones, el nodo x adquiere su nuevo valor del campo *sid* que contiene el mensaje *INF* que se retira del canal. Hay que comprobar que este valor de $s_z.sim\_id_x$ cumple los consecuentes de la propiedad. La hipótesis inductiva establece que el mensaje *INF* cumple la propiedad en s_{z-1} y el consecuente *C2-1* indica que la identidad simulada de ningún nodo puede coincidir con $m.sid$ (donde $\beta = \epsilon$). Al retirarse el mensaje *INF* del canal, $s_z.sim\_id_x$ pasa a valer precisamente $m.sid$, verificándose el consecuente *C1-1*.

  El consecuente *C1-2* también se cumple en s_z. Si no fuera así en s_{z-1} existirían dos mensajes *INF* con $m.sid = (id, t, \gamma)$ y el consecuente *C2-2* establece que no puede haber otro mensaje con el mismo campo *sid* siendo $\beta = \epsilon$.

  Para probar que el efecto estudiado verifica el consecuente *C1-3*, se va a suponer que no se cumple en s_z tras la ejecución de la acción. Según esto, siendo $\gamma = \alpha{\cdot}\beta$, existe un nodo cuya identidad simulada es (id, t, α) que cumple que la variable *cont* asociada es inferior o igual a $first(\beta)$. Por otro lado, la hipótesis inductiva indica que el mensaje *INF* que se retira del canal cumple la propiedad en s_{z-1}. Si $m.sid = (id, t, \gamma)$, de acuerdo al consecuente *C2-3*, existe un nodo k cuya identidad simulada es (id, t, α), siendo $\gamma = \alpha{\cdot}\delta$, que cumple que $s_{z-1}.cont_k > first(\delta)$. Como $s_z.sim\_id_x = m.sid$ y $\gamma = \alpha{\cdot}\delta$, se asume que $s_z.cont_x > first(\delta)$ y, por tanto el consecuente *C1-3* no es falso en s_z.

  A continuación, se supone que el valor adquirido de $s_z.sim\_id_x$ no verifica el consecuente *C2-1*, es decir, $s_z.sim\_id_x = (id, t, \gamma{\cdot}\beta)$. En el efecto analizado el valor de la identidad simulada del nodo x se adquiere del campo *sid* del mensaje *INF* que se retira del canal, es decir, $m.sid = (id, t, \gamma{\cdot}\beta)$. Este mensaje *INF* dirigido al nodo x debe cumplir en s_{z-1} el consecuente *C2-2* y eso resulta imposible de acuerdo al supuesto inicial. Así que, el consecuente *C2-1* también se cumple.

  Considerando que el consecuente *C2-3* no se cumple en s_z, se analiza el efecto de la acción por el que $s_z.sim\_id_x$ adopta el valor del campo *sid* del mensaje *INF* que se retira del canal. Si no se cumple el consecuente *C2-3* se asume que hay un nodo k en s_z cuya identidad simulada es (id, t, α), siendo $\gamma = \alpha{\cdot}\delta$ y, además, $s_z.cont_k \leq first(\delta)$. Ese nodo verificaba también esas condiciones en s_{z-1}. La existencia de un mensaje *INF* con $m.sid = (id, t, \gamma)$ en s_{z-1} y la hipótesis inductiva aseguran el cumplimiento del consecuente *C2-1*. Según este consecuente, en s_{z-1} no hay nigún nodo cuya identidad simulada sea $(id, t,$

$\gamma{\cdot}\beta$). Sin embargo, el nodo k cumpliría entonces que $s_{z-1}.cont_k > first(\delta{\cdot}\beta)$ y, al contrario del supuesto de partida, *C2-3* se verificaría en s_z.

Al ejecutarse esta acción, pueden generarse mensajes *INF* cuyo campo *sid* se construye a partir del del valor que contenía $s_{z-1}.sim\_id_x$ o del campo *sid* del mensaje *INF* que se retira del canal. Si el campo *sid* de los de mensajes *INF* proviene de la identidad simulada de un nodo, la propiedad se verifica del mismo modo que en las acciones $StarDelArc_x(y, t)$ y $Abort_x$. En el caso de que el campo *sid* de un mensaje *INF* proceda del campo *sid* de otro *INF* hay que comprobar que se cumplen el consecuente *C1-2* y los asociados al antecedente *A2* de la propiedad.

Si no se cumpliera *C1-2* en s_z, el campo *sid* del nuevo mensaje sería igual a (id, t, γ). Sabiendo que en s_{z-1}, $m.sid = (id, t, \gamma)$, el campo *sid* del mensaje *INF* que se crea será (id, t, $\gamma{\cdot}c$). Esto implica que $s_z.sim\_id_x$ contiene la terna (id, t, γ) que no es posible por el cumplimiento de *C2-1*.

Para probar que se cumple los consecuentes asociado al antecedente *A2*, se asumirá que en s_z no se verifican y, considerando el efecto de la acción, se alcanzará una contradicción. Si no se cumple *C2-1* en s_z, esto es, existe un nodo cuya identidad simulada sea (id, t, $\gamma{\cdot}\beta$) y el campo *sid* del mensaje *INF* enviado es (id, t, γ) resulta imposible porque hacia el nodo x se dirigiría un mensaje con (id, t, γ), siendo $\gamma = \delta{\cdot}c$. Por tanto, en s_{z-1} existe un nodo cuya identidad simulada tiene como tercer campo una cadena cuyo prefijo está en un mensaje violando el consecuente *C2-1*.

Suponiendo que *C2-2* es falso en s_z, se asume que el campo *sid* del nuevo mensaje *INF* tiene la forma (id, t, $\gamma{\cdot}\beta$). El campo *sid* de este mensaje se ha obtenido añadiendo el contenido de $cont_x$ al campo *sid* del mensaje *INF* que se retira del canal, esto es, (id, t, $\gamma{\cdot}c$). Así que, $c = first(\beta)$ y no se cumplía *C1-3* para el nodo x. Si existiese un mensaje con (id, t, $\gamma{\cdot}\beta$) en s_{z-1} y se generase un mensaje con (id, t, γ) por efectos de π_z, el mensaje que recibiría el nodo x volvería a incluir (id, t, γ) con $\gamma = \delta{\cdot}c$. Esta situación violaría el consecuente *C2-2*

Finalmente, considerando que el consecuente *C2-3* no se verifica en s_z se sabe que la identidad simulada de un nodo k es (id, t, α) y $s_z.cont_k \leq first(\delta)$, siendo $\gamma = \alpha{\cdot}\delta$. Dado que el nodo x adquiere como identidad simulada el campo *sid* de mensaje *INF* que se retira, es decir, $s_z.sim\_id_x = (id, t, \gamma)$, se observa que el campo *sid* del nuevo mensaje es (id, t, $\gamma{\cdot}c$). Siendo $\gamma{\cdot}c = \alpha \cdot \beta$, se llega a la contradicción de que la variable *cont* del nodo x coincide con $first(\beta)$ y eso no es posible por el consecuente *C1-3*.

Por último, otro efecto posible de la acción es la inicialización de $s_z.sim\_id_i$. Esta

variable cambia su valor a $(i,\ s_z.ta_i,\ \epsilon)$ sólo cuando en $s_z.setPredToInf_i$ ya no le quedan elementos. Como $s_{z-1}.satus.id_i = unknown$ (condición de habilitación de la acción), se sabe que el nodo i cambió su tiempo de activación al mismo tiempo que pasaba a ser *unknown*. En ese estado no es posible propagar ni su identidad ni su identidad simulada. Así que, cuando $s_z.sim\_id_i = (i,\ s_z.ta_i,\ \epsilon)$, esta identidad es totalmente nueva en el sistema.

■

La siguiente propiedad ayuda a concretar las caracerísticas que tienen que cumplir las rutas tanto de los mensajes *AVS* y *AVSRSP* como sus versiones almacenadas en set_st_avs y en st_avsrsp, respectivamente. Como estas rutas podrían contar con información incorrecta, la propiedad también considera qué valores de otras variables aseguran que la información no válida se elimina o se recompone para continuar con detección iniciada.

Propiedad B.2.58. Sea $\{n,\ n\prime\} \subseteq \mathcal{N}$, sea $\{t,\ t\prime\} \in \mathbb{N}$, sea $\{b,\ b\prime\} \in \{true,\ false\}$, sea $\{sid,\ sid\prime\} \subseteq T$ y sea $\{p,\ p\prime\} \subseteq P$ tal que $p = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_j,\ t_j)$. Si $((sid,\ t,\ p,\ b) \in s.set\_st\_avs_{n_j} \vee (AVS,\ sid,\ t,\ p,\ b) \in s.channel(n,\ n_j) \vee (t,\ sid,\ p) = s.st\_avsrsp_{n_1} \vee (AVSRSP,\ t,\ sid,\ p) \in s.channel(n,\ n_1)) \wedge \exists\ i$ con $1 < i \leq j$ tal que $(n_{i-1},\ t_{i-1}) \in s.setPred_{n_i} \wedge (n_i,\ t_i) \notin s.setPred_{n_{i+1}}$ entonces:
1. $s.status.alg_{n_1} = candidate \wedge$
2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i,\ t_i) \wedge wait(n_1,\ n_i,\ \gamma_1,\ s) \wedge$
3. $\forall\ l < i$: $s.status.id_{n_l} = known \wedge$
4. $\forall\ l$ tal que $1 < l < i$: $(s.status.alg_{n_l} = dummy \vee (s.status.alg_{n_l} = candidate \wedge sid > s.sim\_id_{n_l} \wedge s.inf\_need_{n_l} = true)) \wedge$
5. $\forall\ l < i$: $s.t\_unk_{n_l} = s.ta_{n_l} \wedge$
6. $(sid,\ t,\ p,\ b) \in s.set\_st\_avs_{n_j} \Rightarrow$
   - $(n_i = n_j \wedge$
   - $(s.status.alg_{n_i} \in \{candidate,\ active,\ victim\} \vee s.status.id_{n_i} = unknown) \wedge$
   - $(((sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.set\_st\_avs_{n\prime} \vee (AVS,\ sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s.channel(n,\ n\prime)) \Rightarrow first(p\prime) \neq (n_1,\ t_1)) \wedge$
   - $(sid = s.sim\_id_{n_1} \wedge s.sim\_id_{n_1} > s.sim\_id_{n_j}) \wedge$
   - $s.nofirstAVS_{n_1} = false \wedge$
   - $s.st\_avsrsp_{n_1} = NULL) \wedge$
7. $(AVS,\ sid,\ t,\ p,\ b) \in s.channel(n,\ n_j) \Rightarrow$

- $((s.status.id_{n_i} = unknown \vee s.status.alg_{n_i} \in \{candidate, active, victim, aborted\} \vee (s.status.alg_{n_i} = dummy \wedge s.ta_{n_i} = t_i \wedge ((b = true \wedge n_i = n_j) \vee (s.blocker_{n_i} = n_{i+1} \wedge n_i \neq n_j)))) \wedge$
- $(((sid\prime, t\prime, p\prime, b\prime) \in s.set\_st\_avs_{n\prime} \vee (AVS, sid\prime, t\prime, p\prime, b\prime) \in s.channel(n, n\prime)) \Rightarrow first(p\prime) \neq (n_1, t_1)) \wedge$
- $sid = s.sim\_id_{n_1} \wedge$
- $s.nofirstAVS_{n_1} = false \wedge$
- $s.st\_avsrsp_{n_1} = NULL) \wedge$

8. $((t, sid, p) = s.st\_avsrsp_{n_1} \vee (AVSRSP, t, sid, p) \in s.channel(n, n_1)) \Rightarrow$
   - $(sid > s.sim\_id_{n_1} \wedge$
   - $((n_i = n_j \wedge (s.status.alg_{n_i} \in \{candidate, active, victim\} \vee s.status\_id_{n_i} = unknown) \wedge s.sim\_id_{n_i} \geq sid) \vee$
   - $((n_i \neq n_j \wedge n_i = n_1) \wedge$
     - $(((INF, sid\prime, s.t\_unk_{n_i}) \in s.channel(n_{i+1}, n_i) \wedge sid\prime > sid) \vee$
     - $((ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF, sid\prime, s.t\_unk_{last(p\prime).id}) \in s.channel\ (n, last(p\prime).id) \wedge sid\prime > sid)))) \vee$
   - $((n_i \neq n_j \wedge n_i \neq n_1) \wedge$
     - $((s.status.alg_{n_i} \in \{candidate, active, victim\} \wedge s.status\_id_{n_i} = known \wedge s.inf\_need_{n_i} = false \wedge s.sim\_id_{n_i} \geq sid) \vee$
     - $((s.status\_id_{n_i} = unknown \wedge s.inf\_need_{n_i} = true) \vee (s.status.alg_{n_i} = candidate \wedge s.inf\_need_{n_i} = true \wedge s.blocker_{n_i} = n_{i+1}) \wedge$
       1. $((sid, t, p - first(p)) = s.st\_alg_{n_1} \wedge$
          - $((((INF, sid\prime, s.t\_unk_{n_i}) \in s.channel(n_{i+1}, n_i) \wedge sid \leq s.sim\_id_{n_i}) \vee$
          - $(ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF, sid\prime, s.t\_unk_{last(p\prime).id}) \in s.channel\ (n, last(p\prime).id) \wedge sid \leq s.sim\_id_{n_i}))) \vee$
       2. $((sid, t, p - first(p)) \neq s.st\_alg_{n_1} \wedge$
          - $(((INF, sid\prime, s.t\_unk_{n_i}) \in s.channel(n_i, n_{i+1}) \wedge sid\prime > sid > s.sim\_id_{n_i}) \vee$
          - $(ewait(n_i, last(p\prime).id, p\prime, s) \wedge (INF, sid\prime, s.t\_unk_{last(p\prime).id}) \in s.channel\ (n, last(p\prime).id) \wedge sid\prime > sid > s.sim\_id_{n_i}))))) \vee$

- $((s.status.alg_{n_i} = dummy \wedge s.blocker_{n_i} = n_{i+1}) \vee (s.status.id_{n_i} = unknown \wedge s.inf\_need_{n_i} = false) \wedge$
  1. $(((sid,\ t,\ p - first(p)) = s.st\_alg_{n_1} \wedge$
     - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
     - $(ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF, sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel\ (n,\ last(p\prime).id) \wedge$
       - $((last(p).id \in nodes(p\prime) \wedge s.sim\_id_{last(p).id} \geq sid \wedge \forall\ (x,\ t_x) \in (visited\_nodes(p\prime) - last(p)) \wedge s.inf\_need_x = false \wedge s.inf\_need_{last(p).id} = true) \vee$
       - $(last(p).id \notin nodes(p\prime) \wedge sid\prime > sid \wedge \forall\ (x,\ t_x) \in visited\_nodes(\mathrm{p}\prime) \wedge s.inf\_need_x = false))))) \vee$
  2. $((sid,\ t,\ p - first(p)) \neq s.st\_alg_{n_1} \wedge$
     - $(((INF,\ sid\prime,\ s.t\_unk_{n_i}) \in s.channel(n_{i+1},\ n_i) \wedge sid\prime > sid) \vee$
     - $((ewait(n_i,\ last(p\prime).id,\ p\prime,\ s) \wedge (INF,\ sid\prime,\ s.t\_unk_{last(p\prime).id}) \in s.channel\ (n,\ last(p\prime).id) \wedge sid\prime > sid))))))))))$

Demostración: En el estado inicial, s_0, $\forall\ \{i,\ j,\ k\} \subseteq \mathcal{N}$ se cumple que $s_0.setPred_i = \emptyset$ por lo que la propiedad es cierta.

- $\pi_z \in \{StartAddArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Para que la acción esté habilitada es necesario que $s_{z-1}.state_x = active$, o por la propiedad B.3.3, que $s_{z-1}.status.alg_x = active$. Este estado impide que el nodo x aparezca en la ruta p en posiciones anteriores al nodo n_i porque debería ser *candidate* o *dummy*. Así que hay que estudiar los casos en los que el nodo x es precisamente el nodo n_i o el nodo x aparece en la ruta a partir del nodo n_i.

  Cuando el nodo x ocupa posiciones posteriores al nodo n_i, se debe analizar la posibilidad de que el nodo x esté incluido en el camino $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$. Si el nodo pertenece a ese camino, $s_{z-1}.status.id_x = unknown$. Al ejecutarse la acción $s_z.status.alg_x$ pasa a ser *blocked*, pero $s_z.status.id_x$ sigue siendo *unknown*. Como las características de las rutas consideradas en la propiedad no se modifican, la propiedad se verifica en s_z. A continuación, se analizan cada una de las ubicaciones de la ruta p que se citan en la propiedad cuando $x = n_i$.

  - Si la ruta p está almacenada en $s_{z-1}.set\_st\_avs_{n_i}$, se sabe que $last(p).id$ coincide con el nodo n_i por la propiedad B.2.21. Además, aplicando la propiedad B.2.16, se tiene que $penult(p) \in s_z.setPred_{n_i}$. Dado que los efectos de la acción modifican la variable $status.alg_{n_i}$, el nodo n_i puede alcanzar los estados *blocked* y *candidate*. Si $s_z.status.alg_{n_i} = candidate$, la propiedad sigue verificándose para el nodo n_i. En el caso de que $s_{z-1}.status.id_{n_i} =$

unknown, los efectos de la acción no modificarán esta variable. Aunque $s_z.status.alg_{n_i} = blocked$, como $s_z.status.id_{n_i} = unknown$ la propiedad se cumple igualmente. Sin embargo, cuando $s_{z-1}.setPredToInf_{n_i} = \emptyset$, resulta que $s_z.status.alg_{n_i} = blocked$ y $s_z.status.id_{n_i} = known$. Considerando la propiedad B.2.47 se obtiene que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$ entonces $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPredToInf_{n_i}$. Como $x = n_i$, se llega a la conclusión de que $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$ y, por tanto, el efecto que transforma $status.alg_{n_i}$ en *blocked* no se puede producir. El cambio de la variable $nofirstAVS_{n_i}$ no afecta porque no es posible que el primer nodo de la ruta sea el nodo n_i por la condición de habilitación.

- Si la ruta p está contenida en un mensaje *AVS* dirigido al nodo n_j, es posible que $n_i \neq n_j$ o $n_i = n_j$, pero sólo tiene sentido analizar el caso en que $x = n_i$. Aplicando la propiedad B.2.47 al nodo n_i, se obtiene que $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$ y, por tanto, no es posible el efecto que convierte $s_z.status.alg_{n_i}$ en *blocked* siendo $s_z.status.id_{n_i} = known$. Los cambios a *candidate* o a *blocked* con $s_z.status.id_{n_i} = unknown$ no afectan y la propiedad sigue verificándose para el nodo n_i en s_z.

  Al ejecutarse la acción, $s_z.blocker_{n_i} = y$, pero la condición sobre la variable $blocker_{n_i}$ sólo se impone cuando $n_i \neq n_j$ y $s_z.status.alg_{n_i} = dummy$. Dado que los efectos de la acción no permiten alcanzar el estado *dummy*, el bloqueo del nodo n_i no influye en el cumplimiento de la propiedad. El cambio de la variable $nofirstAVS_{n_i}$ no afecta al cumplimiento de la propiedad porque el nodo n_i no puede ser el nodo n_1 de acuerdo con la precondición de la acción.

- Si la ruta p está almacenada en $s_{z-1}st\_avsrsp_{n_1}$ o aparece en un mensaje *AVSRSP* que va hacia el nodo n_1, se estudia el único caso de interés para la propiedad que tiene lugar cuando $n_i \neq n_j$ y $n_i \neq n_1$. Los efectos de la acción que cambian el estado del nodo n_i a *candidate* o *blocked* con $s_z.status.id_{n_i} = unknown$ no afectan al cumplimiento de la propiedad porque son modificaciones admitidas cuando $n_i = n_j$. La ejecución de la acción que hace que $s_z.status.alg_{n_i} = blocked$ y $s_z.status.id_{n_i} = known$ no es posible porque requiere que $s_{z-1}.setPredToInf_{n_i} = \emptyset$ y el antecedente asegura lo contrario.

  En el caso en que $n_i \neq n_j$ y la acción esté habilitada, se concluye que la propiedad también se cumple por hipótesis inductiva y porque los estados alcanzados, tras la ejecución de la acción, están permitidos. Por otro lado, para que el efecto de la acción consistente en que $s_z.blocker_{n_i} = y$ pudiera afectar al cumplimiento de la propiedad, $s_z.status.alg_{n_i}$ debería ser *dummy* o *candidate*. Como ese estado no es alcanzable en el supuesto que se estudia $(s_{z-1}.status.id_{n_i} = unknown)$, se asegura que la propiedad es cierta en s_z.

- $\pi_z \in \{EndAddArc_x(y,\, t)\colon \{x,\, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. El único efecto de la acción que afecta a las variables de la propiedad es el que modifica la variable $setPred_x$. Al ejecutarse la acción, el par $(y,\, t)$ se incorpora a $setPred_x$. Este hecho podría hacer que alguna ruta existente en s_{z-1} cumpliera la propiedad en s_z. Esto implicaría que en s_{z-1} el nodo n_i fuera el nodo y y, tras la ejecución de la acción, el nodo n_i que marca la propiedad fuera el nodo x.

  Para que el par $(y,\, t)$ aparezca en un ruta de las indicadas en la propiedad junto con, al menos, otro elemento es necesario que se haya ejecutado previamente las acción $rcvALG_y(x,\, m)$ o $firstAVS_y$. En ambos casos, se debe verificar que $(y,\, t)$ ya perteneciera a $setPred_x$ o, por la propiedad B.3.2, $(y,\, t,\, received)$ estuviera incluido en $set\_waiters_x$. Por la propiedad B.2.1, se deduce que $t \leq ta_y$ en ese instante. Aplicando la propiedad B.3.1, se tiene que $t \leq t\_activ_y$ en ese mismo instante. Por otra parte, la acción está habilitada según el medio cuando la tupla $(y,\, t,\, sent)$ pertenece a $set\_waiters_x$, pero la propiedad B.1.5 asegura que entonces $(y,\, t,\, received)$ no está incluido en $set\_waiters_x$. Así que, si la acción debe estar habilitada será preciso que anteriormente se hayan ejecutado las acciones $StartDelArc_x(y,\, t)$, $EndDelArc_y(x)$ y $StartAddArc_y(x)$. Como la acción $EndDelArc_y(x)$ incrementa el tiempo de activación del nodo y, se concluye que es imposible que $(y,\, t)$ estuviera ya en una ruta en la posición i que indica la propiedad y, posteriormente se incorpore a $setPred_x$ con el mismo tiempo. En consecuencia, la ejecución de esta acción no afecta al cumplimiento de la propiedad.

- $\pi_z \in \{StartDelArc_x(y,\, t)\colon \{x,\, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Una de las posibles condiciones de habilitación de esta acción consiste en que $(y,\, t,\, received) \in s_{z-1}.set\_waiters_x$ y $s_{z-1}.state_y = aborted$ o lo que es lo mismo, aplicando las propiedades B.3.2 y B.3.5 respectivamente, $(y,\, t) \in s_{z-1}.setPred_x$ y $s_{z-1}.status.alg_y = aborted$.

  Considerando el antecedente de la propiedad, se llega a la conclusión de que, si el nodo y pertenece a una de las rutas analizadas entonces $x = n_i$ y $n_i \neq n_1$ y esta situación es imposible para todo nodo anterior a n_i porque su estado debería ser *dummy* o *candidate*. Por otro lado, el nodo x tampoco puede ocupar en las rutas posiciones posteriores al nodo n_i porque entonces el nodo y también formaría parte del camino *ewait*. Dado que la propiedad B.2.15 asegura que un nodo en estado *aborted* nunca puede ser *unknown*, resulta evidente que este supuesto no es posible.

  Al analizar la otra condición de habilitación de la acción, $(y,\, t) \in s_{z-1}.setPred_x$ y $s_{z-1}.status.alg_x = active$, debe comprobarse que el efecto de formación de mensajes *AVS* no afecta al cumplimiento de la propiedad. Estos mensajes se generan a partir de los almacenados en $s_{z-1}.set\_st\_avs_x$ que cumplan que el penúltimo elemento de su ruta coincide con el par $(y,\, t)$. Así mismo, estas rutas

cumplen la propiedad en s_{z-1} por hipótesis inductiva. Los nuevos mensajes *AVS* adoptan como ruta las rutas almacenadas en $s_{z-1}.set\_st\_avs_x$ excepto su último par y se elige como destino de los mismos el nodo que aparece en última posición de su ruta, esto es, el penúltimo nodo de la ruta usada de $s_{z-1}.set\_st\_avs_x$. La estructura de la ruta del mensaje *AVS* permite confirmar que se cumplen todas las características exigidas en la propiedad para los nodos anteriores a n_i. Esto es así porque la ejecución de la acción tiene como efecto indirecto que el nodo y pasa a ser el nodo n_i en s_z para el mensaje *AVS* y no se modifican las variables *status.alg*, *status.id* y *ta* de ningún nodo.

El resto de requisitos que indica la propiedad hacen referencia al nodo n_i. Como el nodo al que va dirigido el mensaje *AVS* es precisamente el nodo y, que coincide con n_i y con n_j, se sabe que $s_z.status.alg_y \in \{dummy, candidate\}$ (en s_{z-1} el nodo y se correspondía con un nodo n_l de los de la propiedad). Si $s_z.status.alg_y = dummy$ además se debe verificar que $s_z.ta_y = t_y$ ya que el último campo del mensaje *AVS* contiene el valor *true*. Dado que el nodo y no cambia su tiempo de activación y estaba incluido en una ruta válida en s_{z-1}, este requisito se cumple. Por otro lado, se comprueba fácilmente que el campo *sid* del mensaje *AVS* es el mismo que el que acompañaba a la ruta almacenada en $s_{z-1}.set\_st\_avs_x$ y éste cumplía en s_{z-1} que era equivalente a $s_{z-1}.sim\_id_{n_1}$. Así que, es obvio que $sid = s_z.sim\_id_{n_1}$ porque no se producen cambios. El resto de características que tienen que ver con el primer nodo de la ruta del mensaje se demuestran aplicando la hipótesis inductiva.

En lo que resta se va a estudiar cómo afecta, en las rutas de interés y que existen en s_{z-1}, el efecto de la acción por el que se retira el par $(y,\ t)$ del conjunto $setPred_x$. Sólo si $s_{z-1}.status.id_x = known$, hay que analizar las consecuencias del efecto considerado porque siendo *unknown* no se modifica ninguna variable de las que aparecen en la propiedad.

- En una ruta almacenada en $s_{z-1}.set\_st\_avs_x$. Al ejecutarse la acción $(y,\ t) \notin s_z.setPred_x$ y, al mismo tiempo, se elimina la ruta de $s_{z-1}.set\_st\_avs_x$. En consecuencia, la propiedad pasaría a verificarse con el antecedente falso. La posible transformación del mensaje almacenado en un mensaje *AVS* ya ha sido demostrada y también cumple la propiedad. Por otra parte, las rutas que siguen almacenadas tras la ejecución de la acción verifican la propiedad por hipótesis inductiva.
- En una ruta incluida en un mensaje *AVS* con destino n_j. Los efectos de la acción no retiran el mensaje del canal y se mantiene el valor del campo *sid*. Si $n_i = x$, al ejecutarse la acción, resulta que $(y,\ t) \notin s_z.setPred_{n_i}$ y el nodo n_i que indica la propiedad pasa a ser el nodo y. Como en s_{z-1} el

mensaje *AVS* cumplía la propiedad, por hipótesis inductiva, se sabe que el estado del nodo y era *dummy* o *candidate*. De acuerdo a lo comentado, en caso de que $n_i = n_j$ en s_{z-1}, tras la ejecución de la acción, el nodo n_i dejará de coincidir con n_j. Es obvio que la propiedad se cumple en esa situación cuando $status.alg_y$ es *candidate*, incluso si el nodo y ocupaba la primera posición de la ruta. Para el caso en el que $status.alg_y$ era *dummy* es preciso asegurar que, como ese estado se mantiene tras la ejecuciń de la acción, $s_z.ta_y = t$ y $s_z.blocker_y = x$. Para comprobar esto hay que recurrir a la condición de habilitación que exigía que $(y,\ t) \in s_{z-1}.setPred_x$ y $s_{z-1}.status.alg_x = active$. A partir de esa condición se deduce que $s_{z-1}.ta_y = t$ y $s_{z-1}.blocker_y = x$. Como los efectos de la acción en este supuesto conservan los valores de estas variables, se concluye que la propiedad también se cumple cuando $s_z.status.alg_y = dummy$. Por último, considerando que los nodos n_i y n_j no coinciden en el mensaje *AVS* existente en s_{z-1} y aplicando los argumentos anteriores, se llega a las mismas conclusiones. La propiedad se verifica igualmente cuando el nodo y no se corresponde con el nodo n_{i-1} de la ruta del mensaje *AVS*. En esa situación los efectos de la acción no afectan al cumplimiento de la propiedad y la hipótesis inductiva concluye.

- En una ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o contenida en un mensaje *AVSRSP* que viaja hacia el nodo n_1. En las condiciones en las que se da el efecto analizado, $s_{z-1}.status.alg_x = active$ y $s_{z-1}.status.id_x = known$, la propiedad B.2.34 asegura que $s_{z-1}.inf\_need_x = false$. La hipótesis inductiva permite demostrar fácilmente que $sid > s_z.sim\_id_{n_1}$. Si $x = n_i$ y la acción se ejecuta, en la nueva ruta $n_i \neq n_j$. Como el nodo y ocupaba una posición anterior al nodo n_i en s_{z-1}, resulta que $s_{z-1}.status.alg_y \in \{dummy,\ candidate\}$ según la propiedad. En ambos casos, $(y,\ t) \in s_{z-1}.setPredToInf_x$ (propiedad B.2.47) y el efecto de la acción en el que $(y,\ t) \notin s_z.setPred_x$ va acompañado por la generación de un mensaje *INF* dirigido al nodo y con $m_{INF}.sid > s_{z-1}.sim\_id_x$. A continuación, se estudian los dos posibles estados del nodo y y se comprueban las características que deben verificarse en cada uno de ellos:
  - Si $s_z.status.alg_y = dummy$ y $s_z.blocker_y = x$, $m_{INF}.sid \geq sid$ tanto para el caso en el que $(sid,\ p - first(p))$ sea equivalente o no a $s_z.st\_alg_{n_1}$. Dado que en s_{z-1} se sabe que $s_{z-1}.sim\_id_x \geq sid$ y el campo sid del mensaje *INF* directo que se envía al nodo y es mayor que $s_{z-1}.sim\_id_x$, resulta evidente que $m_{INF}.sid > sid$ combinando ambas relaciones.
  - Si $s_z.status.alg_y = candidate$, $s_z.inf\_need_y = true$ y $s_z.blocker_y = x$, sólo es posible que $(sid,\ p - first(p)) \neq s_z.st\_alg_{n_1}$. Esto es debido a que, si$sid,\ p - first(p)) = s_z.st_alg_{n_1}$, las rutas son prácticamente iguales y

se puede afirmar que en s_{z-1} el nodo n_i era el mismo en ambas. La propiedad B.2.55 indica que los nodos n_l que ocupan posiciones anteriores al nodo n_i sólo pueden ser *dummy*. En consecuencia, $s_{z-1}.status.alg_y = dummy$ y los efectos de la acción no cambian el estado del nodo y. El supuesto de partida se contradice y se concluye que este caso no es posible. Para el caso en que $(sid,\ p - first(p)) \neq s_z.st\_alg_{n_1}$ hay que comprobar que la siguiente relación: $m_{INF}.sid > sid > s_z.sim\_id_y$. Como en s_{z-1} se cumple que $s_{z-1}.sim\_id_x \geq sid$ y al formarse el mensaje *INF* directo del nodo x al nodo y $m_{INF}.sid > s_{z-1}.sim\_id_x$, se obtiene que $m_{INF}.sid > sid$. Por otro lado, el nodo y verificaba que, por ser un nodo que aparecía en la ruta antes que el nodo n_i en s_{z-1}, $sid > s_{z-1}.sim\_id_y$. Al ejecutarse la acción sim_id_y mantiene su valor y el nodo y pasa a ser el nuevo nodo n_i. Así que, $sid > s_z.sim\_id_y$ como se quería demostrar.

Cuando el nodo x ocupa posiciones posteriores al nodo n_i, el nodo x está incluido en un camino $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$. La definición de camino *ewait* establece que $s_{z-1}.status.id_x = unknown$. Al ejecutarse la acción, el camino *ewait* no se ve modificado y el resto de las variables que intervienen en la propiedad tampoco.

Finalmente, hay que comentar un efecto de la acción por el que $s_z.sim\_id_x$ pasa a valer $(x,\ s_z.ta_x,\ \epsilon)$. Además, cuando se produce este efecto $s_z.setPred_x = \emptyset$. Esto implica que el nodo x no puede formar parte, en ninguna posición, de las rutas consideradas en la propiedad existente.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Al ejecutarse esta acción cambia tanto el tiempo de activación del nodo x como $status.alg_x$ que pasa a *active*. También otro de los efectos de la acción hace que $s_z.st\_avsrsp_x = NULL$. En esa situación el antecedente podría llegar a ser falso.

  La condición de habilitación de la acción según el medio establece que $(x,\ t,\ released) \in s_{z-1}.set\_waiters_y$ o $s_{z-1}.status.alg_y = aborted$, siendo en ambos casos $s_{z-1}.blocker_x = y$. En el primer caso, aplicando la propiedad B.1.5 resulta evidente que $(x,\ t,\ received) \notin s_{z-1}.set\_waiters_y$. De acuerdo con la propiedad B.3.2, eso equivale a que el par $(x,\ t)$ no está incluido en $s_{z-1}.setPred_y$.

  Considerando la otra posibilidad de habilitación, se sabe que $s_{z-1}.setPred_y = \emptyset$ y, en consecuencia, $(x,\ t) \notin s_{z-1}.setPred_y$. Como los pares dispuestos en posiciones previas al nodo n_i para las rutas citadas en la propiedad no pueden cumplir el requisito de habilitación, el nodo x sólo puede ocupar la posición del nodo n_i o la de nodos a partir de n_i.

Suponiendo que el nodo x coincide n_i se van a analizar los efectos de la acción dependiendo de los valores de $s_{z-1}.status.alg_{n_i}$ que afectan a la propiedad.

- $s_{z-1}.status.alg_{n_i} = dummy$: la hipótesis inductiva permite descartar la posibilidad de que el nodo n_i esté contenido en una ruta de $s_{z-1}.set\_st\_avs_{n_i}$ siendo *dummy* su $status.alg_{n_i}$. En caso de que el nodo n_i sea un nodo de la ruta de un mensaje *AVS* dirigido al nodo n_j en s_{z-1} se sabe que la ejecución de la acción modifica el estado del nodo n_i de manera que $s_z.status.alg_{n_i} = active$ y $s_z.blocker_{n_i}$ pase a ser *NULL*. Dado que este nuevo estado está entre los permitidos en la propiedad, la ruta del mensaje *AVS* la sigue haciendo cierta en s_z. Del mismo modo, es posible que el nodo n_i aparezca en la ruta de un mensaje *AVSRSP* con destino el nodo n_1 o almacenado en $s_{z-1}.st\_avsrsp_{n_1}$, donde $n_i \neq n_1$. Si las rutas mencionadas cumplen la propiedad en s_{z-1} y $s_{z-1}.status.alg_{n_i} = dummy$, la propiedad B.2.34 señala que $s_{z-1}.inf\_need_{n_i} = false$. Dado que el nodo x se corresponde con el nodo n_i, la propiedad B.2.47 confirma que $s_{z-1}.setPredToInf_{n_i} \neq \emptyset$. En esta situación resulta que $s_z.status.id_{n_i} = unknown$ y $s_z.inf\_need_{n_i} = false$. De acuerdo con los valores de esas variables, cuando el nodo n_i alcanza el estado *active* tras la ejecución de la acción, la propiedad sigue cumpliéndose.
- $s_{z-1}.status.alg_{n_i} = candidate$: cuando la ruta que contiene al nodo n_i está almacenada en $s_{z-1}.set\_st\_avs_{n_i}$, los efectos de la acción no afectan al cumplimiento de la propiedad a pesar de que $s_z.status.alg_{n_i} = active$ y $s_z.status.id_{n_i}$ sea *known* o *unknown*. Así mismo, la ejecución de la acción no afecta, en ningún caso, a la ruta de un mensaje *AVS* que se dirija al nodo n_j y que incluya al nodo n_i en s_{z-1}. Si el nodo n_i se localiza en una ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o incluida en un mensaje *AVSRSP* que va hacia el nodo n_1, se pueden distinguir distintas situaciones según coincida el nodo n_i con el nodo n_j o no. Suponiendo que $n_i = n_j$, la hipótesis inductiva asegura que $s_{z-1}.sim\_id_{n_i} \geq sid$. Al ejecutarse la acción, se sigue cumpliendo esa condición y $s_z.status.alg_{n_i} = active$. Por consiguiente, la propiedad se verifica. En el caso de que $n_i \neq n_j$ y $s_{z-1}.inf\_need_{n_i} = false$, la ejecución de la acción mantiene el valor de las variables $inf\_need_{n_i}$, $status.id_{n_i}$ y $sim\_id_{n_i}$. Por todo ello, la propiedad se cumple en s_z. Para $s_{z-1}.inf\_need_{n_i} = true$, los efectos de la acción convierten $status.id_{n_i}$ en *unknown* sin modificar el valor de $inf\_need_{n_i}$. El requisito exigido en s_{z-1} sobre la variable $blocker_{n_i}$ deja de ser preciso en s_z y, por tanto, la propiedad se verifica de igual manera que en s_{z-1}.
- $s_{z-1}.status.alg_{n_i} = blocked$: aplicando la hipótesis inductiva, se deduce que sólo es posible que $s_{z-1}.status.id_{n_i} = unknown$. Como esta variable no

cambia su valor tras la ejecución de la acción, una ruta almacenada en $s_{z-1}.set\_st\_avs_{n_i}$ o incluida en un mensaje *AVS* dirigido al nodo n_j y que contenga al nodo n_i sigue cumpliendo la propiedad en s_z. De forma similar, para una ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o la de un mensaje *AVSRSP* con destino el nodo n_1 en la que aparece el nodo n_i, la propiedad es cierta ya sea $n_i = n_j$ o $n_i \neq n_j$. Al ejecutarse la acción, $s_z.status.alg_{n_i} = active$, pero se mantiene el valor correspondiente de $inf\_need_{n_i}$ y $s_z.status.id_{n_i} = unknown$. Esto implica que la propiedad se verificará en s_z cumpliendo las mismas condiciones que en s_{z-1}.

En el caso de que el nodo x forme parte de un camino $ewait(n_i, last(p\prime).id, p\prime, s_{z-1})$, siendo $x \neq n_i$, la definición de *ewait* asegura que $s_{z-1}.status.id_x = unknown$. Al ejecutarse la acción, el camino *ewait* no se modifica y el resto de las variables que intervienen en la propiedad tampoco.

- $\pi_z \in \{Abort_x\colon x \in \mathcal{N}\}$. Para que la acción esté habilitada es necesario que $s_{z-1}.status.alg_x = victim$. Esta condición reduce los casos de estudio y hace que sea sólo sea posible que el nodo x coincida con el nodo n_i. En posiciones anteriores al nodo n_i los nodos deben ser *dummy* o *candidate* y a partir del nodo n_i, el nodo x formaría parte de un camino *ewait* y entonces $s_{z-1}.status.id_x = unknown$. La propiedad B.2.15 asegura que al estado *victim* le correspondería $s_{z-1}.status.id_x = known$. A continuación se repasan las posibilidades de que la acción se ejecute y existan rutas de interés que ya existieran en s_{z-1}.

  En el supuesto de que haya almacenada una ruta en $s_{z-1}.set\_st\_avs_x$, la ejecución de la acción puede hacer falso el antecedente de la propiedad porque se eliminan todos los elementos de ese conjunto.

  Un efecto de esta acción consiste en la formación de mensajes *AVS* a partir de los almacenados en $s_{z-1}.set\_st\_avs_x$ cuando éstos existen. Estos nuevos mensajes también cumplen la propiedad. Dado que los mensajes almacenados en $s_{z-1}.set\_st\_avs_x$ verifican la propiedad por hipótesis inductiva, es fácil mostrar que los mensajes *AVS* generados cumplen que:

  1. $s_z.status.alg_{n_1} = candidate$. La ruta del mensaje *AVS* coincide totalmente con la correspondiente al mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ salvo el último elemento de esta última que se elimina. Por tanto, como el primer elemento es el mismo y no se modifica el estado de ese nodo resulta que es *candidate*.
  2. $\exists\ n_i = y\colon p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = n_i$ tal que $wait(n_1, n_i, \gamma_1, s_z)$. Como la ruta del mensaje *AVS* se obtiene eliminando el último elemento de la ruta del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$, el nodo n_i que indica la

propiedad cambia al ejecutarse la acción porque el par $(y,\ t) \notin s_z.setPred_x$. En resumen, tras los efectos de la acción el nodo n_i deja de ser el nodo x y pasa a ser el nodo y. Como las características de la ruta hasta el nodo y no se modifican este requisito queda demostrado por hipótesis inductiva.

3. $\forall\ l < i,\ s_z.status.id_{n_l} = known$. Dado que la ruta del mensaje *AVS* se construye a partir de la ruta almacenada $s_{z-1}.set\_st\_avs_x$ y el estado de los nodos de esta última previos al nodo n_i eran *known*, se deduce que el estado de todos ellos se conserva incluso siendo $n_i = y$.
4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$.
5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$.
6. Esta acción no modifica ni *status.alg* ni *status.id* del nuevo nodo n_i. Como en s_{z-1} el nuevo nodo i verificaba la propiedad siendo un nodo n_l de los de la propiedad, se tiene que $s_z.status.alg_y \in \{dummy,\ candidate\}$ lo cual verifica los requisitos de la propiedad porque $m.fwd = true \wedge n_i = n_j$. Si el nodo n_i es *dummy*, resulta obvio que cumple que $s_z.ta_{n_i} = t_i$ porque este nodo en s_{z-1} pertenecía a un camino *wait* los efectos no modifican su tiempo de activación.
7. Dado que $n.sid = m.sid$, $s_z.sim\_id_{first(m.path).id} = s_{z-1}.sim\_id_{first(n.path).id}$ y $s_{z-1}.sim\_id_{first(n.path).id} = n.sid$, es obvio que $m.sid = s_z.sim\_id_{first(m.path).id} = s_z.sim\_id_{n_1}$.
8. $\forall\ m\prime \in s_z.set\_st\_avs_{n_k} \vee m\prime \in s_z.channel(n,\ n_k) \wedge m\prime.tipo = AVS \Rightarrow first(m\prime.path) \neq (n_1,\ t_1)$. Aplicando la hipótesis inductiva también se demuestra que $sid = s_z.sim\_id_{n_1}$, $s_z.nofirstAVS_{n_1} = false$ y $s_z.st\_avsrsp_{n_1} = NULL$. Basta con tener en cuenta que *first(m.path).id* = *first(n.path).id* = n_1 y que los efectos de la acción no afectan a ese nodo.

Analizando otros efectos que tienen que ver con la existencia de las rutas indicadas en la propiedad, se aprecia que la acción está habilitada cuando $s_{z-1}.status.alg_x = victim$. Esta condición obliga a que sean motivo de estudio los casos en que $x = n_i$ o en los que el nodo x ocupe posiciones posteriores a la del nodo n_i en la ruta. En el resto de posiciones, el estado de los nodos debe ser *dummy* o *candidate*.

1. *ruta almacenada en* $\mathrm{s}_{z-1}.\mathrm{set\_st\_avs}_x$. Si $i = 1$, como $n_i = n_j = x$, se tiene un mensaje almacenado con un único elemento. La propiedad indica que $s_{z-1}.status.alg_x = candidate$, pero se requiere que $s_{z-1}.status.alg_x = victim$ para que la acción esté habilitada. Si $i \neq 1$ y $n_2 \neq x$, al ejecutar la acción $Abort_x$, el mensaje de $s_{z-1}.set\_st\_avs_x$ se elimina y se transforma en

un mensaje *AVS*. Ya se ha analizado que este nuevo mensaje cumple la propiedad en s_z. Sin embargo, cuando $n_2 = x$, el mensaje *AVS* no se llega a enviar e igualmente se elimina la ruta almacenada en $s_{z-1}.set\_st\_avs_x$.

2. *ruta contenida en un mensaje* AVS *con destino* n_j.

   - Si $i \neq 1$ y $n_i = n_j = x$, los efectos de la acción no retiran ningún mensaje de este tipo y tampoco modifican su estructura. La precondición de la acción establece que $s_{z-1}.status.alg_x = victim$ y, al ejecutarse la misma, resulta que $s_z.setPred_x = \emptyset \wedge s_z.status.alg_x = aborted$. Aunque el estado alcanzado es uno de los permitidos en la propiedad, el nodo x ya no es n_i en s_z. El nuevo n_i coincide con el elemento anterior a x en la ruta, es decir, uno de los nodos que en s_{z-1} cumplía los requisitos de los nodos n_l. Como $s_{z-1}.status.alg_{n_l} \in \{dummy, candidate\}$ es obvio que el nuevo nodo n_i verifica la propiedad porque $n_i \neq n_j$. Si $s_z.status.alg_{n_i} = dummy$ es evidente que $s_z.ta_{n_i} = t_i \wedge s_z.blocker_{n_i} = n_{i+1}$ porque en el estado previo este nodo pertenecía a un camino *wait* y $s_{z-1}.ta_{n_{i-1}} = s_z.ta_{n_i} \wedge s_{z-1}.blocker_{n_{i-1}} = s_z.blocker_{n_i}$, siendo $n_{i+1} = x$.
     En el caso en que, además, $n_2 = x$, al ejecutar la acción $Abort_x$, el mensaje *AVS* sigue verificando la propiedad porque $n_i \neq n_j$, $n_1 = n_i$ y $s_z.status.alg_{n_1} = s_{z-1}.status.alg_{n_1} = candidate$ que es un estado permitido por al propiedad.
   - Si $i = 1$, como $n_i = n_j = x$, se tiene un mensaje almacenado con un único elemento. La propiedad indica que $s_{z-1}.status.alg_x = candidate$, pero se requiere que $s_{z-1}.status.alg_x = victim$ para que la acción esté habilitada.
   - Considerando que $n_i \neq n_j$ e $i = 1$, se llega a la conclusión de que la acción no podrá estar habilitada, si en la ruta se cumplía que $s_{z-1}.status.alg_x = candidate$. La acción $Abort_x$ requiere que $s_{z-1}.status.alg_x = victim$.
   - Cuando $i \neq 1$, $n_i \neq n_j$, la ejecución de la acción cambia el nodo n_i, pero como ya se ha explicado, el mensaje *AVS* sigue cumpliendo la propiedad. Para $n_2 = x$, resulta que n_1 pasa a ser el nodo n_i de la propiedad y se mantiene que $n_i \neq n_j$. Como $s_z.status.alg_{n_1} = candidate$, queda demostrado que el mensaje *AVS* verifica la propiedad.

3. *ruta almacenada en* $\mathrm{s}_{z-1}.\mathrm{st\_avsrsp}_{n_1}$ *o enviada en un mensaje* AVSRSP *al nodo* n_1. En el análisis de este caso se considera que $x = n_i$ y se comprueba fácilmente que:

   a) $sid > s_z.sim\_id_{n_1}$ porque la ruta no se modifica y $s_{z-1}.sim\_id_{n_1} = s_z.sim\_id_{n_1}$.

b) Si $n_i = n_j$ y $i = 1$, se sabe que $s_{z-1}.status.alg_{n_1} = candidate$ (consecuente cierto) y la acción queda habilitada sólo cuando $s_{z-1}.status.alg_{n_1} = victim$.

c) Para $n_i = n_j$, $i \neq 1$ y $n_2 \neq x$, se tiene que la ejecución de la acción hace que $s_z.status.alg_x = aborted$ pero $n_i \neq x$ porque, al mismo tiempo, $s_z.setPred_x = \emptyset$. En consecuencia, la ruta que se analiza en s_z cumple que $n_i \neq n_j$. Además, eso conlleva que en s_z hay un nuevo nodo n_i que cumple la propiedad y que en el estado anterior formaba parte de uno de los nodos referenciados como n_l. Como $s_{z-1}.status.alg_{n_l} \in \{dummy, candidate\}$ y *status.alg* del nuevo n_i no cambia, se tiene que:

- $s_z.status.alg_{n_i} = dummy \wedge s_z.blocker_{n_i} = x$. En estas circunstancias es preciso que un mensaje de tipo *INF* viaje desde el nodo x hasta el nodo n_i en s_z. Confirmando que $m_{INF}.sid > sid$ se podrá concluir que la propiedad es cierta tanto en este caso. El mensaje *INF* que se pone en el canal por los efectos de la acción cumple que $m_{INF}.sid = (s_{z-1}.sim\_id_x.id,\ s_{z-1}.sim\_id_x.ta,\ s_{z-1}.sim\_id_x.nu.cont)$.
  Considerando la relación de orden $s_{z-1}.sim\_id_x < m_{INF}.sid \wedge s_{z-1}.sim\_id_x \geq sid$, se llega a la conclusión de que $sid < m_{INF}.sid$, tal y como se quería demostrar. Esto es válido tanto para $(sid,\ p - first(p)) = s_z.st\_alg_{n_1}$ como para $(sid,\ p - first(p)) \neq s_z.st\_alg_{n_1}$.
- $s_z.status.alg_{n_i} = candidate \wedge s_z.blocker_{n_i} = x$. El nuevo nodo n_i verificaba en s_{z-1} que $sid > s_{z-1}.sim\_id_{n_l} \wedge s_{z-1}.inf\_need_{n_l} = true$. Se pueden plantear las siguientes situaciones:
  - $(sid,\ p - first(p)) = s_z.st\_alg_{n_1}$. La propiedad B.2.55 establece que el nuevo nodo n_i sólo puede ser *dummy*. Así que este caso es imposible.
  - $(sid,\ p - first(p)) \neq s_z.st\_alg_{n_1}$. En el canal debe haber un mensaje de tipo *INF* con destino n_i y origen x tal que: $m_{INF}.sid > sid \wedge sid > s_z.sim\_id_{n_i}$. La primera condición, queda confirmada al comprobar que $m_{INF}.sid = (s_{z-1}.sim\_id_x.id,\ s_{z-1}.sim\_id_x.ta,\ s_{z-1}.sim\_id_x.nu.cont) \wedge s_{z-1}.sim\_id_x < m_{INF}.sid$ (relación de orden) $\wedge\ s_{z-1}.sim\_id_x \geq sid$ (hipótesis inductiva). Por todo ello, $m_{INF}.sid > sid$. Este mensaje *INF* que se menciona aparece por los efectos de la acción. Por otra parte, el último requisito se cumple porque la acción no modifica estas variables para un nodo n_l de la ruta y $sid > s_z.sim\_id_{n_i}$.

d) Si $n_i = n_j$, $i \neq 1$ y $n_2 = x$, la ruta sigue cumpliendo la propiedad porque $n_i = n_j$ e $i = 1$. Los efectos de la acción $Abort_x$ ponen de

manifiesto la existencia de un mensaje *INF* en el que $m_{INF}.sid > sid$. Basta con recordar que $m_{INF}.sid > s_{z-1}.sim\_id_x$ y que en s_{z-1}, con $n_i = n_j = x$, se cumplía que $s_{z-1}.sim\_id_x \geq sid$.

e) $n_i \neq n_j$ e $i = 1$. Si la ruta verifica la propiedad en s_{z-1} con el consecuente cierto, la acción no estará habilitada porque es necesario que $s_{z-1}.status.alg_x = victim$.

f) Con $n_i \neq n_j$, $i \neq 1$ y $n_2 \neq x$, se deben estudiar dos posibilidades. Tras la ejecución de la acción, en s_z hay un nuevo nodo n_i que cumple la propiedad y que en el estado anterior formaba parte de uno de los nodos referenciados como n_l. Como $s_{z-1}.status.alg_{n_l} \in \{dummy, candidate\}$ y *status.alg* del nuevo n_i no cambia, se tiene que:

- $s_z.status.alg_{n_i} = dummy \wedge s_z.blocker_{n_i} = x$. Tanto para $(sid, p - first(p)) = s_z.st\_alg_{n_1}$ como para $(sid, p - first(p)) \neq s_z.st\_alg_{n_1}$, el mensaje *INF* que se pone en el canal por los efectos de la acción cumple que $m_{INF}.sid = (s_{z-1}.sim\_id_x.id, s_{z-1}.sim\_id_x.ta, s_{z-1}.sim\_id_x.nu.cont)$.
  Considerando la relación de orden, $s_{z-1}.sim\_id_x < m_{INF}.sid$, y $s_{z-1}.sim\_id_x \geq sid$, se llega a la conclusión de que $sid < m_{INF}.sid$, tal y como se impone en las dos situaciones que indica la propiedad.
- $s_z.status.alg_{n_i} = candidate \wedge s_z.inf\_need_{n_i} = true \wedge s_z.blocker_{n_i} = x$. Teniendo en cuenta las características de nuevo nodo n_i resulta evidente que no es posible que $(sid, p - first(p)) = s_z.st\_alg_{n_1}$. La propiedad B.2.55 confirma que el estado del nuevo nodo n_i sólo puede ser *dummy*. En s_{z-1} es evidente que tanto la ruta p que se analiza como $s_{z-1}.st\_alg_{n_1}.path$ poseen el mismo nodo n_i. Esto reduce el análisis al caso $(sid, p - first(p)) \neq s_z.st\_alg_{n_1}$. Por tanto, en el canal debe haber un mensaje de tipo *INF* con destino n_i y origen x tal que: $m_{INF}.sid \geq sid \wedge sid > s_z.sim\_id_{n_i}$. Este mensaje *INF* es un efecto de la acción $Abort_x$. La primera condición, queda confirmada al comprobar que $m_{INF}.sid = (s_{z-1}.sim\_id_x.id, s_{z-1}.sim\_id_x.ta, s_{z-1}.sim\_id_x.nu.cont) \wedge s_{z-1}.sim\_id_x < m_{INF}.sid$ (relación de orden) $\wedge\ s_{z-1}.sim\_id_x \geq sid$ (hipótesis inductiva). Por todo ello, $m_{INF}.sid > sid$. Por otra parte, el último requisito se cumple porque la acción no modifica estas variables para un nodo n_l de la ruta y $sid > s_z.sim\_id_{n_i}$.

g) Cuando $n_i \neq n_j$, $i \neq 1$ y $n_2 = x$, la ejecución de la acción conlleva la creación de un mensaje *INF* y n_i pasa a ser el nodo n_1. Ese mensaje *INF* cumple que $m_{INF}.sid > sid$ porque $m_{INF}.sid > s_{z-1}.sim\_id_x$ y $s_{z-1}.sim\_id_x \geq sid$.

Para que se generen mensajes *INF* es necesario que $s_{z-1}.setPredToInf_x \neq \emptyset$. Los mensajes tienen como origen el nodo x y como destino el nodo id ya que $(id,\ t) \in s_{z-1}.setPredToInf_x$.

En s_{z-1} se cumple que existe $wait(n_1,\ n_i,\ \gamma_1,\ s_{z-1})$, siendo $n_i = x$ lo que implica que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_x$. Aplicando la propiedad B.2.47, se obtiene que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPredToInf_x$. Así que, éste será el nodo al que se envíe el mensaje *INF*.

Caso $x \in p\prime$ tal que $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$:

La definición de *ewait* obliga a que $s_{z-1}.status.id_x = unknown$, pero la condición de habilitación exige que $s_{z-1}.status.alg_x = victim$ o, según la propiedad B.2.15, $s_{z-1}.status.id_x = known$.

Otro efecto de la acción modifica sim_id_x de manera que $s_z.sim\_id_x = (x,\ s_z.ta_x,\ \epsilon)$. Se puede apreciar que esto sucede cuando $s_z.setPred_x = \emptyset$. En esta situación, por tanto, el nodo x no puede ser ya el n_i de ninguna ruta existente (antecedente falso).

- $\pi_z \in \{initiate_x\colon\ x \in \mathcal{N}\}$. Los efectos de la acción no introducen cambios en el antecedente de la propiedad. Además, la acción está deshabilitada para los nodos que ocupan posiciones anteriores al nodo n_i en las posibles rutas. La precondición de la acción indica que $s_{z-1}.status_x = blocked$ y esos nodos o bien son *dummy* o bien son *candidate*. Del mismo modo, si el nodo x ocupa posiciones posteriores al nodo n_i y forma parte de un camino $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$, según la definición de *ewait*, $s_{z-1}.status.id_x$ debería ser *unknown* y la condición de habilitación de la acción exige que sea *known*.

  Considerando que el nodo x coincide con el nodo n_i y que la ruta p está contenida en $s_{z-1}.set\_st\_avs_{n_j}$, en un mensaje *AVS* dirigido al nodo n_j, en $s_{z-1}.st\_avsrsp_{n_1}$ o en un mensaje *AVSRSP* con destino el nodo n_1, se observa que, entre los estados permitidos en la propiedad para el nodo n_i, no se admite que $s_{z-1}.status.alg_{n_i}$ sea *blocked* ni $s_{z-1}.status.id_{n_i}$ sea *known*.

- $\pi_z \in \{rcvALG_x(y,m)\colon\ \{x,\ y\} \subseteq \mathcal{N} \wedge \wedge\ m \in M_{ALG}\}$. Al ejecutar esta acción, puede llegar a almacenarse un mensaje en $s_z.st\_avsrsp_x$. Hay que comprobar, por tanto, que ese mensaje cumple todas las condiciones de la propiedad:

  1. $s_z.status.alg_{n_1} = candidate$. Este efecto requiere que $s_{z-1}.status.alg_x = candidate$ y esta variable no cambia al ejecutarse la acción. Además, el primer nodo de la ruta almacenada en $s_z.st\_avsrsp_x$ coincide con el nodo x. Así que, es obvio que se cumple este requisito.

2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i, t_i)$. La ruta que se almacena en $s_z.st\_avsrsp_x$ está formada por el par $(x, s_z.ta_x)$ y la ruta del mensaje ALG que se retira del canal como efecto de la acción. Este mensaje ALG cumple la propiedad B.2.55 en s_{z-1}. Por lo tanto, $\exists$ una posición i en su ruta tal que $\forall\ l < i$ se verifica que $t_l = s_z.ta_{n_l}$ y $(n_l, t_l) \in s_z.setPred_{n_{l+1}}$. La propiedad B.2.55 también asegura que $(x, s_z.ta_x) \in s_z.setPred_{n_1}$, siendo n_1 el primer nodo de la ruta del mensaje ALG. Por todo ello, se cumple la condición impuesta.
3. $\forall\ l < i$, $s_z.status.id_{n_l} = known$. Como $s_z.status.alg_x = candidate$ y el nodo x es el primer nodo de la ruta almacenada, la propiedad B.2.15 permite afirmar que $s_z.status.id_{n_1} = known$. Para el resto de nodos de la ruta, hay que tener en cuenta que, según la propiedad B.2.55, $\forall\ l < i$ se cumple que $s_z.status.alg_{n_l} = dummy$. La propiedad B.2.15 asegura entonces que $status.id_{n_l}$ para todos ellos es $dummy$, verificando así el requisito.
4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Para todos esos nodos se puede afirmar, gracias a la propiedad B.2.55, que $s_z.status.alg_{n_l} = dummy$.
5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Dado que $\forall\ l$ tal que $1 < l < i$ se sabe que $s_z.status.alg_{n_l} = dummy$ y la propiedad B.2.29 señala que en ese estado se verifica que $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$, la condición se cumple para esos nodos. Sin embargo, para el primer nodo de la ruta almacenada hay que tener en cuenta que $s_z.status.alg_x = candidate$ y que $s_z.st\_alg_x \neq NULL$ por los efectos de esta misma acción. Asumiendo esa situación, se aplica de nuevo la propiedad B.2.29 y se concluye que también $s_z.t\_unk_x = s_z.ta_x$.

A continuación se comprueban otros requisitos que debe cumplir el mensaje almacenado en $s_z.st\_avsrsp_x$.

- $sid > s_z.sim\_id_{n_1}$. El almacenamiento del mensaje ALG en $s_z.st\_avsrsp_x$ se produce cuando el valor del campo sid del mensaje ALG es superior al de $s_{z-1}.sim\_id_x$. Dado que el primer nodo de la ruta almacenada coincide con el nodo x y se adopta como campo sid precisamente el del mensaje ALG, se cumple la característica porque sim_id_x no varía.
- Si el nodo n_i coincide con el nodo n_j en la ruta del mensaje ALG, se tienen que estudiar los estados posibles del nodo n_i. Cuando se produce el efecto que se analiza siendo $s_{z-1}.status.alg_{n_i} \in \{candidate, active, victim\}$, no cambia el estado del nodo n_i, pero hay que comprobar que $sid \geq s_z.sim\_id_{n_i}$. Como el valor de $sim\_id_{n_i}$ tampoco se modifica tras la ejecución de la acción y la propiedad B.2.56 indica que, en cualquiera de los estados del supuesto, $s_{z-1}.sim\_id_{n_i} \geq m_{ALG}.sid$, se concluye que $s_z.sim\_id_{n_i} \geq sid$.

Esto resulta evidente porque, en el almacenamiento del mensaje ALG en $s_z.st\_avsrsp_x$, se adopta como campo *sid* el del mensaje ALG.

Otra posibilidad que hay que tener en cuenta es que $s_{z-1}.status.id_{n_i} = unknown$ y $s_{z-1}.inf\_need_{n_i} = true$. La propiedad B.2.56 establece que $s_{z-1}.sim\_id_{n_i} \geq m_{ALG}.sid$. Dado que la variable $sim\_id_{n_i}$ no cambia al ejecutarse la acción en el efecto analizado, se obtiene que $s_z.sim\_id_{n_i} \geq sid$ porque *sid* coincide con $m_{ALG}.sid$. En consecuencia, la propiedad se cumple.

Si $s_{z-1}.status.id_{n_i} = unknown$ pero $s_{z-1}.inf\_need_{n_i} = false$, hay que deducir el estado del nodo n_i en el que se incorporó a la ruta y conocer las acciones que permiten que ocupe la última posición de la ruta que pasa a almacenarse en $s_z.st\_avsrsp_x$. Para ello, el estado del nodo n_i debió ser inicialmente *candidate*. La única acción que permitiría que $s_{z-1}.status.id_{n_i}$ llegara a ser posteriormente *unknown* es la acción $EndDelArc_{n_i}(n)$. La propiedad B.2.50 asegura que el nodo n_i, que era *candidate* cuando empezó a formarse la ruta p, no puede convertirse en *dummy* si se mantiene inalterado su correspondiente $setPred_{n_i}$. Esto implica que el nodo n_i antes de ser ejecutada la acción $EndDelArc_{n_i}(n)$ seguía siendo *candidate* aunque el valor de $inf\_need_{n_i}$ pudiera tomar tanto el valor *true* como *false*. Realmente, que $inf\_need_{n_i}$ sea *false* impide que se alcance el estado $s_{z-1}.status.id_{n_i} = unknown$ porque los efectos de la acción $EndDelArc_{n_i}(n)$ hacen que $status.id_{n_i}$ permanezca como *known*. Aunque el nodo n_i volviera a bloquearse, el estado seguiría siendo *known* porque $setPredToInf_{n_i} \neq \emptyset$.

En cambio, cuando $inf\_need_{n_i} = true$ y se ejecuta $EndDelArc_{n_i}(n)$, los efectos de esta acción provocan que $status.id_{n_i} = unknown$. Como el caso que se estudia precisa que $s_{z-1}.inf\_need_{n_i} = false$, es obvio que se debe recibir un mensaje *INF* para conseguir ese valor. Las acciones que convierten el estado del nodo n_i en *victim* también pueden transformar $inf\_need_{n_i}$ en *false*. Sin embargo, en ese estado el nodo n_i solamente tendría habilitada la acción $Abort_{n_i}$ y, en consecuencia, el nodo pasaría a ser *aborted*. Alcanzado ese estado, el nodo n_i ya no puede cambiar de estado de nuevo. Así que, la recepción de un mensaje *INF* permitiría que $inf\_need_{n_i}$ fuera *false*, pero el estado del nodo n_i también se vería modificado y pasaría a ser *candidate* o *active* con $status.id_{n_i} = known$ porque $setPredToInf_{n_i} \neq \emptyset$ según el antecedente de la propiedad. En consecuencia, siendo en cualquiera de esos estados imposible alcanzar que $s_{z-1}.status.id_{n_i} = unknown$ y, a la vez, $s_{z-1}.inf\_need_{n_i} = false$.

Suponiendo que en la ruta del mensaje ALG $n_i \neq n_j$ se pueden considerar los siguientes casos según el estado del nodo n_i:

- Si $s_{z-1}.status.alg_{n_i} \in \{candidate, active, victim\}$, $s_{z-1}.status\_id_{n_i} = known$ y $s_{z-1}.inf\_need_{n_i} = false$, la propiedad B.2.56 indica que en los estados citados $s_{z-1}.sim\_id_{n_i} \geq m_{ALG}.sid$. Como los efectos de la acción no modifican la variable $sim\_id_{n_i}$ y al componer el mensaje que se almacena en $s_z.st\_avsrsp_x$ se incluye tal cual el campo *sid* del mensaje *ALG*, se concluye que $s_z.sim\_id_{n_i} \geq sid$ como exige la propiedad.
- Si $s_{z-1}.status\_id_{n_i} = unknown$ y $s_{z-1}.inf\_need_{n_i} = true$, la propiedad B.2.56 señala que $s_{z-1}.sim\_id_{n_i} \geq m_{ALG}.sid$, cumpliéndose además que existe un mensaje *INF* directo al nodo n_i o indirecto a través de un camino *ewait* que empieza en n_i. Al ejecutarse la acción, dado que el efecto analizado provoca que $(sid,\ p - first(p)) = s_z.st\_alg_x$, el mensaje *INF* existente se mantendrá en el canal y $s_z.sim\_id_{n_i} \geq sid$ ya que el campo *sid* almacenado coincide con $m_{ALG}.sid$.
- Si $s_{z-1}.status.alg_{n_i} = candidate$, $s_{z-1}.inf\_need_{n_i} = true$ y $s_{z-1}.blocker_{n_i} = n_{i+1}$ se recurre a la propiedad B.2.54 para demostrar la existencia en el canal de un mensaje *INF* dirigido al nodo n_i de forma directa o indirecta. Para ello, se observa que se cumplen las condiciones del antecedente *A2* para n_i:
  - $s_{z-1}.st\_alg_{n_i} \neq NULL$. Si $s_{z-1}.inf\_need_{n_i} = true$, la propiedad B.2.35 lo asegura.
  - $s_{z-1}.t\_unk_{n_i} = s_{z-1}.ta_{n_i}$. La propiedad B.2.29 confirma que la condición se cumple porque $s_{z-1}.status.alg_{n_i} = candidate$ y $s_{z-1}.st\_alg_{n_i} \neq NULL$.
  - $s_{z-1}.blocker_{n_i} = n_j$. Si $s_{z-1}.status.alg_{n_i} = candidate$, la propiedad B.3.4 establece que $s_{z-1}.state_{n_i} = blocked$ y, de acuerdo con la propiedad del medio B.1.1, $s_{z-1}.blocker_{n_i} \neq NULL$. Como la acción analizada no modifica la variable $blocker_{n_i}$ es evidente que $n_j = n_{i+1}$.

  Según sea el valor de $s_{z-1}.status.id_{n_{i+1}}$ se cuenta con dos posibilidades. Si es *known*, la propiedad B.2.4 afirma que $(n_i,\ t_i) \notin s_{z-1}.setPredToInf_{n_{i+1}}$. Esto implica que se verifica el consecuente *C2-3* de la propiedad B.2.54 y, por tanto, existe un mensaje *INF* hacia el nodo n_i cuyo campo *sid* es superior al que está almacenado en $s_{z-1}.st\_alg_{n_i}$. En caso de que $s_{z-1}.status.id_{n_{i+1}} = unknown$, como $(n_i,\ t_i) \notin s_{z-1}.setPred_{n_{i+1}}$, se tiene que sólo es posible que $(n_i,\ s_{z-1}.ta_{n_i},\ released) \in s_{z-1}.set\_waiters_{n_{i+1}}$ por la propiedad B.3.2. De acuerdo con la propiedad B.2.39 podría pensarse que $s_{z-1}.status.alg_{n_{i+1}} = aborted$, pero al pertenecer el nodo n_{i+1} a un camino *ewait*, se sabe que $s_{z-1}.status.id_{n_{i+1}} = unknown$ y la propiedad B.2.15 asegura que el valor de la variable *status.id* es *known* siendo *aborted*. En consecuencia, el consecuente *C2-5* de la propiedad B.2.54 se cumple en s_{z-1}, esto es, existe un mensaje *INF* que alcanzará el último nodo de un camino *ewait* que

comienza precisamente en el nodo n_i.

Una vez verificada la existencia de un mensaje *INF* en s_{z-1}, sólo queda demostrar que $sid \leq s_z.sim\_id_{n_i}$. La propiedad B.2.56 cuando $s_{z-1}.status.alg_{n_i} = candidate$ indica que $s_z.sim\_id_{n_i} = m_{ALG}.sid$. Como se adopta el mismo campo *sid* del mensaje *ALG* para el mensaje que se almacena en $s_z.st\_avsrsp_x$ y $status.alg_{n_i}$ no varía, se llega a la conclusión de que $s_z.sim\_id_{n_i} \geq sid$.

- Si ($s_{z-1}.status.alg_{n_i} = dummy$ y $s_{z-1}.blocker_{n_i} = n_{i+1}$) o ($s_{z-1}.status.id_{n_i} = unknown$ y $s_{z-1}.inf\_need_{n_i} = true$). El mensaje *ALG* que existe en el canal en s_{z-1} verificando los estados indicados para el nodo n_i cumple la propiedad B.2.56 por hipótesis inductiva. De acuerdo a esta propiedad, en s_{z-1} existe en el canal un mensaje *INF* dirigido al nodo n_i directa o indirectamente a través de un camino *ewait* encabezado por n_i. El valor del campo *sid* de ese mensaje *INF* es superior al del mensaje *ALG* en s_{z-1} y, al ejecutarse la acción, esa característica se mantiene. Como el campo *sid* del mensaje *ALG* es el que se adopta para almacenarse en $s_z.st\_avsrsp_x$ y sólo se tiene en cuenta la posibilidad de que la ruta almacenada en $s_z.st\_avsrsp_x$ coincida, salvo el primer elemento, con la almacenada en $s_z.st\_alg_{n_1}$, la propiedad se verifica. Si el mensaje *INF* es indirecto, la propiedad queda demostrada teniendo en cuenta que $s_{z-1}.sim\_id_{last(p).id} \geq m_{ALG}.sid$ y el campo *sid* almacenado en $s_z.st\_avsrsp_x$ es precisamente $m_{ALG}.sid$. En esta situación sólo tiene sentido considerar que $last(p).id \in p\prime$.

Otro de los posibles efectos de la acción puede hacer que $s_z.status.alg_x = victim$. Seguidamente se analiza si este cambio de estado del nodo x afecta al cumplimiento de la propiedad de cualquier mensaje de los indicados en la propiedad en los que estuviera presente en s_{z-1}.

En primer lugar, se van a considerar las rutas en las que el nodo x aparece en ellas en posiciones anteriores al nodo n_i. Si la ruta está contenida en $s_{z-1}.set\_st\_avs_{n_i}$, todos los nodos desde x hasta el nodo n_i están incluidos en los que forman parte del ciclo recogido en la ruta del mensaje *ALG*. Como la propiedad B.2.55 establece que el nodo n_i es un nodo cuyo estado es *dummy*, la hipótesis inductiva indica que el nodo n_i no puede ser *dummy*. Por tanto, se concluye que no es posible que haya almacenado un mensaje en $s_z.set\_st\_avs_{n_i}$ y sea precisamente el nodo x el que pasa a *victim* debido a la recepción de un mensaje *ALG*.

Para la ruta contenida en un mensaje *AVS* hacia el nodo n_j, la hipótesis inductiva indica que, para todo nodo n_l anterior al nodo n_i,$s_{z-1}.t\_unk_x = s_{z-1}.ta_x$. A partir de esta igualdad y aplicando la propiedad B.2.46, se llega a la conclusión

de que, en las condiciones indicadas, no puede existir un mensaje *ALG* que señale al nodo x como *victim* y, a la vez, exista otro mensaje *AVS* en el que el nodo x sea un nodo anterior al nodo n_i.

Si la ruta está almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o está incluida en un mensaje *AVSRSP* con destino el nodo n_1, la propiedad B.2.55, junto con las características del modelo *SR*, indican que el nodo n_i forma parte de la ruta que contiene el mensaje *ALG*. Por tanto, la propiedad B.2.55 señala que $s_{z-1}.status.alg_{n_i} = dummy$. El único caso de la propiedad en el que este estado es posible es cuando $n_i \neq n_j$ y $n_i \neq n_1$. En ese estado, es necesario que se cumpla que $s_{z-1}.blocker_{n_i} = n_{i+1}$. Además, el nodo n_i cumple que $s_{z-1}.blocker_{n_i} = n_k$ y $(n_i, t_i) \in s_{z-1}.setPred_{n_k}$. De todo esto, se deduce que $n_i = n_k$, pero el antecedente de la propiedad requiere que $(n_i, t_i) \notin s_{z-1}.setPred_{n_{i+1}}$. En conclusión, resulta imposible que haya un mensaje de tipo *ALG* que señale como víctima al nodo x y, al mismo tiempo, este nodo esté contenido en la ruta de un mensaje *AVSRSP* o almacenado en *st_avsrsp* siendo el nodo x un nodo anterior al nodo n_i.

Cuando el nodo x coincide con el nodo n_i identificado en la propiedad y éste está contenido en una ruta almacenada en $s_{z-1}.set\_st\_avs_{n_i}$ o en una ruta incluida en cualquier mensaje *AVS* dirigido al nodo n_j, el cambio de estado no influye en el cumplimiento de la propiedad para esas rutas ya que el estado *victim* es un estado posible. Si se considera una ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o de un mensaje *AVSRSP* con destino el nodo n_1, el efecto que hace que $s_z.status.alg_{n_i} = victim$ no afecta en los casos en que el nodo n_i se corresponda con el nodo n_j o en el que siendo nodos diferentes y $s_{z-1}.inf\_need_{n_i} = false$. El supuesto en el que $n_i \neq n_j$, $s_{z-1}.inf\_need_{n_i} = true$ y $s_{z-1}.blocker_{n_i} = n_{i+1}$ no es posible porque el nodo que recibe el mensaje *ALG*, según la propiedad B.2.55, debe pertenecer a $s_{z-1}.setPred_y$ y además $s_{z-1}.blocker_{n_i} = y$. Como el antecedente de la propiedad requiere que $(n_i, t_i) \notin s_{z-1}.setPred_{n_{i+1}}$. Del mismo modo, tampoco es posible que siendo $n_i \neq n_j$, el nodo n_i sea precisamente el nodo n_1. Como el nodo n_i forma parte del ciclo existente en la ruta del mensaje *ALG*, el nodo n_i debería pertenecer al conjunto $setPred_{n_2}$ y eso contradice el antecedente de la propiedad.

En el caso en que el nodo x forme parte de la ruta en posiciones posteriores al nodo n_i de la propiedad, se sabe que la acción no estará habilitada porque en esa ubicación el nodo x pertenece a un camino *ewait*. En ese tipo de camino la variable $s_{z-1}.status.id_x = unknown$ y la habilitación de la acción precisa que $s_{z-1}.status.id_x = known$.

- $\pi_z \in \{firstAVS_x\colon x \in \mathcal{N}\}$. Al ejecutarse esta acción se genera un mensaje *AVS* que debe cumplir la propiedad. Una de las precondiciones de la acción indica que $s_{z-1}.cand\_succ_x \neq NULL$. La propiedad B.2.17 asegura que en esta situación

$s_{z-1}.status.alg_x = candidate$. Como los efectos de esta acción no cambian esta variable, se tiene que $s_z.status.alg_x = candidate$. Observando la estructura del nuevo mensaje, se sabe que el primer elemento de su ruta es $(x,\ s_z.ta_x)$ y este par va seguido $s_z.st\_alg_x.path$. Además, el valor del campo *sid* del mensaje *AVS* es tomado de $s_z.sim\_id_x$. A continuación se muestran otras características que el mensaje *AVS* tiene que verificar:

1. $s_z.status.alg_{n_1} = candidate$. Como $first(\mathrm{p}).id = x$, resulta obvio dado que $s_z.status.alg_x = candidate$.
2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i,\ t_i)$. La propiedad B.2.55 asegura que, en la ruta almacenada en $s_z.st\_alg_x$, si $\exists\ i$ con $1 < i < j$ tal que $(n_{i-1},\ t_{i-1}) \in s_z.setPred_{n_i}$ entonces $\forall\ l < i$: $t_l = s_z.ta_{n_l}$ y $(n_l,\ t_l) \in s_z.setPred_{n_{l+1}}$. La misma propiedad B.2.55 también confirma que $(x,\ s_z.ta_x) \in s_z.setPred_{n_1}$.
3. $\forall\ l < i$, $s_z.status.id_{n_l} = known$. De nuevo, la propiedad B.2.55 confirma que, en $s_z.st\_alg_x.path$, $\forall\ l < i$ $s_z.status.alg_{n_l} = dummy$ y $s_z.status.alg_x = candidate$. Aplicando la propiedad B.2.15 a la ruta del mensaje *AVS*, resulta que $\forall\ l < i$, $s_z.status.id_{n_l} = known$.
4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Considerando la propiedad B.2.55, se concluye que todos los nodos de la ruta almacenada en $s_z.st\_alg_x.path$ son *dummy*.
5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Sabiendo que los nodos de $s_z.st\_alg_x.path$ cumplen que $\forall\ l < i$ $s_z.status.alg_{n_l} = dummy$, la propiedad B.2.29 establece que $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Para el primer nodo de la ruta del mensaje *AVS* también se cumple esta condición, $s_z.t\_unk_x = s_z.ta_x$, porque $s_z.status.alg_x = candidate$ y $s_z.st\_alg_x \neq NULL$ (propiedad B.2.29).

Otros requisitos que debe cumplir el mensaje *AVS* se comentan a continuación:

- Caso $n_i = n_j$: La propiedad B.2.56 determina que, si existe un mensaje almacenado en $s_{z-1}.st\_alg_x$, los posibles estados de n_i son $s_{z-1}.status.alg_{n_i} \in \{candidate, active, dummy, victim\}$ o $s_{z-1}.status.id_{n_i} = unknown$. Los efectos de la acción no modifican ni *status.alg* ni *status.id*. La propiedad analizada permite en s_z todos esos estados a excepción del estado *dummy* cuando el campo *b* del mensaje *AVS* vale *false*. Como el campo *b* del mensaje *AVS* creado por esta acción tiene precisamente ese valor hay que demostrar que no es posible que $s_{z-1}.status.alg_{n_i} = dummy$. Dado que $last(s_{z-1}.st\_alg_x.path).id = n_i$ y observando las acciones que incorporan pares a una ruta, se sabe que el nodo n_i era *candidate* cuando se incorporó y el par $(n_{i-1},\ t_{i-1})$ pertenecía al conjunto $setPred_{n_i}$. En estas circunstancias,

la propiedad B.2.50 asegura que el nodo n_i nunca puede llegar a ser *dummy* porque que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$ tal y como exige el antecedente de la propiedad.

- Caso $n_i \neq n_j$: Al igual que en el caso anterior, la propiedad B.2.56 admite como posibles estados del nodo n_i: $s_{z-1}.status.alg_{n_i} \in \{candidate,\ active,\ dummy,\ victim\}$ o $s_{z-1}.status.id_{n_i} = unknown$.

  En el supuesto de que $s_{z-1}.status.alg_{n_i}$ fuera *dummy*, se debe comprobar que, además, $s_{z-1}.ta_{n_i} = t_i$ y $s_{z-1}.blocker_{n_i} = n_{i+1}$ porque los efectos de la acción no cambian ninguna de esas variables. Suponiendo que $s_{z-1}.ta_{n_i} \neq t_i$, la propiedad B.2.1 señala que $s_{z-1}.ta_{n_i} < t_i$. Esto implica que el nodo n_i después de incorporarse a la ruta ejecutó la acción $EndDelArc_{n_i}(n_{i+1})$ o $Abort_{n_i}$ porque son las únicas acciones que incrementan el tiempo de activación del nodo n_i. De acuerdo con la propiedad B.1.6, se sabe que el nodo n_i que pasa a ser *aborted* tras la ejecución de la acción $Abort_{n_i}$ no puede llegar a ser *dummy* en estados posteriores. Por otra parte, si se ejecuta $EndDelArc_{n_i}(n_{i+1})$ siendo $status.alg_{n_i} = dummy$, la propiedad B.2.11 señala que $s_{z-1}.setPred_{n_i} = s_{z-1}.setPredToInf_{n_i}$. Como $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$, entonces ese par también pertenece a $s_{z-1}.setPredToInf_{n_i}$. Los efectos de $EndDelArc_{n_i}(n_{i+1})$ en estas condiciones hacen que $status.alg_{n_i} = active$ y $status.id_{n_i} = unknown$. En consecuencia, el nodo n_i precisa de un mensaje *INF* para volver a ser *known*. Sin embargo, después de recibirlo, el estado alcanzado sería *candidate*. Cuando se ejecuta $EndDelArc_{n_i}(n_{i+1})$ siendo *candidate* el nodo n_i, la propiedad B.2.50 asegura que, mientras $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$, el estado final del nodo n_i nunca podrá ser *dummy*. De todo lo expuesto, se concluye que sólo es posible que $s_{z-1}.ta_{n_i} = t_i$ siendo $s_{z-1}.status.alg_{n_i} = dummy$ y al ejecutarse la acción se mantendrá esa igualdad y el estado del nodo n_i.

  En lo que respecta a la variable $blocker_{n_i}$, se puede afirmar que $s_{z-1}.blocker_{n_i} = n_{i+1}$ porque, tal y como se ha analizado previamente, $s_z.ta_{n_i} = t_i$ y eso significa que el nodo n_i no se activa y mantiene el valor de la variable $blocker_{n_i}$. La propiedad B.1.1 indica que $s_{z-1}.blocker_{n_i} \neq NULL$ cuando $s_{z-1}.status.alg_{n_i} = dummy$. Para comprobar que $s_{z-1}.blocker_{n_i}$ es precisamente el nodo n_{i+1} basta con recordar que cuando un nodo se incorpora a la ruta, $(n_i,\ t_i) \in s_{z-1}.setPred_{n_{i+1}}$ y según la propiedad B.3.2, $(n_i,\ t_i,\ received) \in s_{z-1}.set\_waiters_{n_{i+1}}$. Finalmente, mediante la propiedad B.1.4, se confirma que $s_{z-1}.blocker_{n_i} = n_{i+1}$.

- $(sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s_z.set\_st\_avs_{n\prime} \vee (AVS,\ sid\prime,\ t\prime,\ p\prime,\ b\prime) \in s_z.channel(n,\ n\prime) \Rightarrow first(p\prime) \neq (n_1,\ t_1)$. Al ejecutarse la acción sólo se genera el mensaje analizado. Si existieran en s_z otros mensajes de interés, éstos deberían

existir en s_{z-1}. De acuerdo con la hipótesis inductiva $s_{z-1}.nofirstAVS_x = false$ y esta condición impide que la acción esté habilitada.

- $sid = s_z.sim\_id_{n_1}$. Es obvio ya que el nodo x coincide con el primer elemento de la ruta del mensaje *AVS*.
- $s_z.nofirstAVS_{n_1} = false$. Los efectos de la acción hacen que se verifique este requisito.
- $s_z.st\_avsrsp_{n_1} = NULL$. Las condiciones de habilitación de la acción aseguran que $s_{z-1}.st\_avsrsp_x = NULL$ (propiedad B.2.51). Tras la ejecución de la acción, $s_{z-1}.st\_avsrsp_x = s_z.st\_avsrsp_x$.

▪ $\pi_z \in \{rcvAVS_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Al retirar del canal el mensaje *AVS* se pueden producir las siguientes situaciones:

- El mensaje *AVS* se transforma en un mensaje tipo *AVSRSP*, $m\prime$, que debe cumplir la propiedad.
- El mensaje *AVS* se retira del canal, siendo $m.fwd = true$ y $s_{z-1}.sim\_id_x \leq m.sid$. En caso de que el mensaje no se almacene porque ya está contenido en $s_{z-1}.set\_st\_avs_x$, la propiedad se cumple por hipótesis inductiva
- Se elimina un mensaje de $s_{z-1}.set\_st\_avs_x$ porque el mensaje m recibido contiene su ruta. Posteriormente, se incorpora el mensaje *AVS* recibido en $s_z.set\_st\_avs_x$.
- El mensaje *AVS* pasa a formar parte de $s_z.set\_st\_avs_x$. La ruta almacenada no se modifica y, por tanto, todos los requisitos que son comunes a los de un mensaje *AVS* se verifican igualmente.
- Se genera un nuevo mensaje *AVS* construido a partir del recibido de manera que su ruta $m\prime.path = m.path - last(m.path)$, siendo $last(m.path) = x$ por la propiedad B.2.21. Según las condiciones de la acción, $penult(m.path) \notin s_{z-1}.setPred_x$ y en consecuencia $x \neq n_i$. Al generar el nuevo mensaje *AVS*, $last(m\prime.path) = penult(m.path)$, podría darse que $last(m\prime.path).id = n_i \vee last(m\prime.path).id \neq n_i$ (ocupando una posición superior a la que marca i).

  En este caso, no se produce ningún cambio de estado los nodos de la ruta y ya se ha visto que el antecedente sigue siendo cierto. Los requisitos que verifica el mensaje *AVS* siguen cumpliéndose en el nuevo mensaje porque atañen a los nodos localizados en posiciones inferiores a la del nodo n_i.
- Otro de los posibles efectos de la acción hace que $s_z.status_x = victim$. En esa situación el mensaje *AVS* es retirado del canal pero no se almacena en $s_z.set\_st\_avs_x$.

Observando las precondiciones de la acción se puede llegar a la conclusión de que la propiedad se cumplía en s_{z-1} con el consecuente cierto.

- el mensaje *AVS* se transforma en un mensaje tipo *AVSRSP*, $m\prime$, que debe cumplir la propiedad. Se puede comprobar fácilmente que:
  1. $s_z.status.alg_{n_1} = candidate$. Sólo hay que tener en cuenta que $n_1 = first(m.path) = first(m\prime.path)$. Por tanto, $s_{z-1}.status.alg_{first(m\prime.path)} = candidate$.
  2. $\exists\ n_i$: $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1).id = n_i$ tal que $wait(n_1, n_i, \gamma_1, s_z)$. Considerando que la propiedad B.2.21, establece que $last(m.path) = x$ y que una de las condiciones del efecto que se analiza es que $penult(m.path) \in s_{z-1}.setPred_x$, se tiene que el nodo x coincide con el nodo n_i al que hace referencia la propiedad. Es por tanto evidente que el nodo n_i de la propiedad para el mensaje *AVS* recibido se mantiene en el mensaje *AVSRSP* que se genera. Como sus rutas son exactamente iguales, no se modifica *ta* ni *setPred* de ningún nodo, $wait(n_1, n_i, \gamma_1, s_{z-1})$ pasa a ser $wait(n_1, n_i, \gamma_1, s_z)$.
  3. $\forall\ l < i\ s_z.status.id_{n_l} = known$. La hipótesis inductiva asegura que esto se cumple en s_{z-1} para los nodos correspondientes de *m.path*. Como $m\prime.path = m.path$ y no se modifica *status.id* de ningún nodo, $\forall\ l < i\ s_z.status.id_{n_l} = known$.
  4. $\forall\ l, 1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Este punto también se verifica porque la ruta de los dos mensajes son iguales. Los efectos de la acción en este caso en particular no provocan ningún cambio de estado.
  5. $\forall\ l, l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Aplicando la hipótesis inductiva, se llega a la conclusión de que los nodos de la ruta de $m\prime.path$ verifican que $\forall\ l\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$.
  6. Al ejecutarse la acción y producirse el efecto analizado se tiene que el mensaje *AVSRSP*, $m\prime$, verifica:
     - $m\prime.sid > s_z.sim\_id_{n_1}$. La estructura del nuevo mensaje indica que $m\prime.sid = s_z.sim\_id_x$ y la condición de este efecto de la acción obliga a que $s_{z-1}.sim\_id_x > m.sid$. Sustituyendo en esta expresión las caracteríticas relacionadas con el campo *sid* del mensaje *AVS*, $m.sid = s_{z-1}.sim\_id_{n_1}$, y de formación del mensaje *AVSRSP*, resulta que $m\prime.sid > s_{z-1}.sim\_id_{n_1}$. Como $s_{z-1}.sim\_id_{n_1} = s_z.sim\_id_{n_1}$ y $s_{z-1}.sim\_id_x = s_z.sim\_id_x$, se obtiene $m\prime.sid > s_z.sim\_id_{n_1}$.
     - En s_{z-1} el mensaje *AVS* verifica que $n_i = n_j = x$. Al formarse el mensaje *AVSRSP* se sigue cumpliendo que $n_i = n_j = x$. Este

mensaje, según la propiedad, debe cumplir que ($s_z.status.alg_{n_i} \in \{candidate, active, victim\} \vee s_z.status\_id_{n_i} = unknown$) y $s_z.sim\_id_{n_i} \geq sid$. Las variables $status.alg_x$, $status.id_x$ en este efecto no se ven modificadas. Por eso, si $s_{z-1}.status.alg_x \in \{candidate, active, victim\} \wedge s_{z-1}.status\_id_x = known$ (precondición de la acción) $\wedge$ $s_{z-1}.inf\_need_x = false$, para que el consecuente sea cierto bastará con demostrar que $s_z.sim\_id_x \geq m\prime.sid$. Es obvio que esta expresión se cumple porque al formarse el mensaje *AVSRSP* se impone que $m\prime.sid = s_z.sim\_id_x$.

- Se elimina un mensaje de $s_{z-1}.set\_st\_avs_x$ porque el mensaje recibido contiene su ruta. Al mismo tiempo, hay un efecto que incorpora la ruta del mensaje *AVS* recibido, *m.path*, en $s_z.set\_st\_avs_x$ como $m\prime.path$. Así que, con $m.fwd = true$ se cumple lo siguiente:

  1. $s_z.status.alg_{n_1}$ =*candidate*. Sólo hay que tener en cuenta que $n_1 = first(m.path) = first(m\prime.path)$ y que $s_{z-1}.status.alg_{first(m.path)} = candidate$. De ahí se desprende que $s_{z-1}.status.alg_{first(m\prime.path)} = candidate$, siendo $m\prime \in s_z.set\_st\_avs_x$.
  2. $\exists\ n_i$: $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1).id = n_i$ tal que $wait(n_1, n_i, \gamma_1, s_z)$. Considerando que la propiedad B.2.21, establece que $last(m.path).id = x$ y que una de las condiciones del efecto que se analiza es que $penult(m.path) \in s_{z-1}.setPred_x$, se tiene que el nodo x coincide con el nodo n_i al que hace referencia la propiedad. Es por tanto evidente que el nodo n_i de la propiedad para el mensaje *AVS* recibido se mantiene en el mensaje $m\prime$ que se almacena en $s_z.set\_st\_avs_x$. Como sus rutas son exactamente iguales, no se modifica *ta* ni *setPred* de ningún nodo, $wait(n_1, n_i, \gamma_1, s_{z-1})$ pasa a ser $wait(n_1, n_i, \gamma_1, s_z)$.
  3. $\forall\ l < i\ s_z.status.id_{n_l} = known$. La hipótesis inductiva asegura que la condición sobre la variable *status.id* se cumple en s_{z-1} para los nodos correspondientes de *m.path*. Como $m\prime.path = m.path$ y no se modifica *status.id* de ningún nodo, $\forall\ l < i\ s_z.status.id_{n_l} = known$.
  4. $\forall\ l, 1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Este punto también se verifica porque la ruta de los dos mensajes son iguales. Los efectos de la acción, en este caso en particular, no provocan ningún cambio de estado.
  5. $\forall\ l, l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Aplicando la hipótesis inductiva, se llega a la conclusión de que los nodos de la ruta de $m\prime.path$ verifican que $\forall\ l\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$.

6. Al almacenarse un mensaje $m\prime$ en $s_z.set\_st\_avs_x$ también se debe verificar que :
   - $n_i = n_j$. Dado que $m\prime.path = m.path$, $last(m\prime.path).id = n_j = x$ (propiedad B.2.21) y como $penult(m.path) \in s_{z-1}.setPred_x$, la propiedad B.2.16 dice que $n_i = x$.
   - $s_z.status.alg_x \in \{candidate, active, victim\} \vee s_z.status.id_x = unknown$. Los posibles estados del nodo x en s_{z-1} para $b = true$, según la hipótesis inductiva, son $s_{z-1}.status.alg_x \in \{candidate, active, victim, dummy, aborted\} \vee s_{z-1}.status.id_x = unknown$. Si $s_{z-1}.status.alg_x = dummy \Rightarrow s_{z-1}.ta_x = t_x$.
     La condición de habilitación de la acción descarta la posibilidad de que $s_{z-1}.status.id_x = unknown$. Además, el efecto estudiado tampoco se podría producir si $s_{z-1}.status.alg_x = aborted$ porque $s_{z-1}.setPred_x = \emptyset$. La hipótesis inductiva confirma que $(n_{i-1}, t_{i-1}) \in s_{z-1}.setPred_x$ y esto se mantiene en s_z. Para $s_{z-1}.status.alg_x = dummy$, como $n_i = n_j$, sólo es posible que el mensaje AVS del canal en s_{z-1} tenga $m.fwd = true$. Además, con este estado, $s_{z-1}.ta_x = t_x$, pero la precondición de la acción exige que $s_{z-1}.ta_x > last(m.path).ta = t_x$. Por todo ello, la acción no está habilitada en este caso.
     En el resto de situaciones, el estado del nodo x del mensaje almacenado será uno de los indicados en la propiedad porque se mantiene el valor de $s_{z-1}.status.alg_x$.
   - $m\prime.sid = s_z.sim\_id_{n_1} \wedge s_z.sim\_id_{n_1} > s_z.sim\_id_x$. En primer lugar, al almacenarse el mensaje se tiene que $m\prime.sid = m.sid$. Como en s_{z-1} se cumplía que $m.sid = s_{z-1}.sim\_id_{n_1}$ y $s_{z-1}.sim\_id_{n_1} = s_z.sim\_id_{n_1}$, queda demostrado que $m\prime.sid = s_z.sim\_id_{n_1}$.
     Por otra parte, este efecto se produce cuando $s_{z-1}.sim\_id_x \leq m.sid$. La ejecución de la acción no modifica las variables de esta desigualdad, $s_z.sim\_id_x \leq m\prime.sid$. Como $m\prime.sid = s_z.sim\_id_{n_1}$, resulta que $s_z.sim\_id_x \leq s_z.sim\_id_{n_1}$. La igualdad de esta expresión no es posible porque la propiedad B.2.57 dice que no existen dos identidades simuladas iguales. Según eso, si las identidades simuladas son iguales entonces, los nodos deben ser iguales $x = n_1$. Considerando que $x = n_1$ se llega a la conclusión de que el efecto de la acción que se consigue es el que convierte en $victim$ al nodo x porque $last(m\prime.path) = last(m.path)$.
   - $s_z.nofirstAVS_{n_1} = false$. La hipótesis inductiva permite afirmar que $s_{z-1}.nofirstAVS_{n_1} = false$. Como los efectos no modifican esta variable, queda comprobada esta característica.

- $s_z.st\_avsrsp_{n_1} = \emptyset$. La hipótesis inductiva permite afirmar que $s_{z-1}.st\_avsrsp_{n_1} = NULL$. Como los efectos no modifican esta variable, queda comprobada este requisito.
- $\forall\ m\prime \in s_z.set\_st\_avs_{n_k} \vee m\prime \in s_z.channel(n, n_k) \wedge m.type = AVS \Rightarrow first(m\prime.path) \neq (n_1, t_1)$. Si existe un mensaje como el descrito, distinto al que se retira del canal, es obvio que $m\prime \in s_z.set\_st\_avs_{n_k} \vee m\prime \in s_{z-1}.channel(n, n_k)$. De acuerdo con la hipótesis inductiva, este mensaje cumplía la propiedad en s_{z-1}. Al ejecutar la acción se mantiene $first(m.path) \neq (n_1, t_1)$, como se quería demostrar.

- El mensaje *AVS* recibido, m, pasa a ser un elemento, $m\prime$, de $s_z.set\_st\_avs_x$. La ruta no sufre ninguna modificación ya sea $m.fwd = true$ o $m.fwd = false$. Es obvio que cumple todos los requisitos que son comunes a un mensaje *AVS* porque se comparten todos los elementos de la ruta. A pesar de ello, es necesario comprobar que:
  - $n_i = n_j$. Dado que $m\prime.path = m.path$, $last(m\prime.path).id = n_j = x$ (propiedad B.2.21) y como $penult(m.path) \in s_{z-1}.setPred_x$, la propiedad B.2.16 dice que $n_i = x$.
  - $s_z.status.alg_x \in \{candidate, active, victim\} \vee s_z.status.id_x = unknown$.
    1. $m.fwd = true$
       Los posibles estados del nodo x en s_{z-1}, según la hipótesis inductiva son $s_{z-1}.status.alg_x \in \{candidate, active, dummy, victim, aborted\} \vee s_{z-1}.status.id_x = unknown$. La condición de habilitación de la acción descarta la posibilidad de que $s_{z-1}.status.id_x = unknown$ porque $s_{z-1}.status.id_x = known$. Además, el efecto estudiado tampoco se podría producir si $s_{z-1}.status.alg_x = aborted$ porque $s_z.setPred_x = \emptyset$.
       Para $s_{z-1}.status.alg_x = dummy$, como $n_i = n_j$, sólo es posible que el mensaje *AVS* del canal en s_{z-1} tenga $m.fwd = true$. Además, con este estado, $s_{z-1}.ta_x = t_x$, pero la precondición de la acción exige que $s_{z-1}.ta_x > last(m.path).ta = t_x$. Por todo ello, la acción no está habilitada en este caso. Esto implica que el estado del nodo x del mensaje almacenado será uno de los indicados en la propiedad.
    2. $m.fwd = false$
       Los posibles estados del nodo x en s_{z-1}, según la hipótesis inductiva son $s_{z-1}.status.alg_x \in \{candidate, active, victim, aborted\} \vee s_{z-1}.status.id_x = unknown$. Sin embargo, la condición de habilitación de la acción descarta alguna de esas opciones porque $s_{z-1}.status.id_x = known$. Además, el efecto estudiado tampoco se podría producir si $s_{z-1}.status.alg_x = aborted$ porque $s_z.setPred_x =$

$\emptyset$. Por tanto, el estado del nodo x del mensaje almacenado será uno de los indicados en la propiedad.

- $\forall\ m\prime \in s_z.set\_st\_avs_{n_k} \vee m\prime \in s_z.channel(n,\ n_k) \wedge m.type = AVS \Rightarrow first(m.path) \neq (n_1,\ t_1)$. Si existe un mensaje como el descrito, distinto al que se retira del canal, es obvio que $m \in s_{z-1}.channel(n,\ x)$. De acuerdo con la hipótesis inductiva, este mensaje cumplía la propiedad en s_{z-1}. Al ejecutar la acción se mantiene $first(m.path) \neq (n_1,\ t_1)$, como se quería demostrar.
- $m\prime.sid = s_z.sim\_id_{n_1} \wedge s_z.sim\_id_{n_1} > s_z.sim\_id_x$. En primer lugar, al almacenarse el mensaje se tiene que $m\prime.sid = m.sid$. Como en s_{z-1} se cumplía que $m.sid = s_{z-1}.sim\_id_{n_1}$ y $s_{z-1}.sim\_id_{n_1} = s_z.sim\_id_{n_1}$, queda demostrado que $m\prime.sid = s_z.sim\_id_{n_1}$.
  Por otra parte, este efecto se produce cuando $s_{z-1}.sim\_id_x \leq m.sid$. La ejecución de la acción no modifica las variables de esta desigualdad, $s_z.sim\_id_x \leq m\prime.sid$. Como $m\prime.sid = s_z.sim\_id_{n_1}$, resulta que $s_z.sim\_id_x \leq s_z.sim\_id_{n_1}$. La igualdad de esta expresión no es posible porque la propiedad B.2.57 dice que no existen dos identidades simuladas iguales. Según eso, si las identidades simuladas son iguales entonces, los nodos deben ser iguales $x = n_1$. Considerando que $x = n_1$ se llega a la conclusión de que el efecto de la acción que se consigue es el que convierte en *victim* al nodo x.
- $s_z.nofirstAVS_{n_1} = false$ y $s_z.st\_avsrsp_{n_1} = NULL$. Como los efectos no modifican estas variables, la hipótesis inductiva concluye.

- Se pone en el canal un nuevo mensaje *AVS*, $m\prime$, construido a partir del recibido de manera que $m\prime.path = m.path - last(m.path)$, siendo $last(m.path) = x$ por la propiedad B.2.21. Según las condiciones de la acción, $penult(m.path) \notin s_{z-1}.setPred_x$ y en consecuencia $x \neq n_i$. Al generar el nuevo mensaje *AVS*, $last(m\prime.path) = penult(m.path)$, podría darse que $last(m\prime.path).id = n_i \vee last(m\prime.path).id \neq n_i$ (ocupando una posición superior a la que marca i).

  En este caso, no se produce ningún cambio de estado los nodos de la ruta y ya se ha visto que el antecedente sigue siendo cierto. Los requisitos que verifica el mensaje *AVS* siguen cumpliéndose en el nuevo mensaje porque atañen a los nodos localizados en posiciones inferiores a la del nodo n_i.

- Otro de los posibles efectos de la acción puede hacer que $s_z.status_x = victim$, pero en esa situación el mensaje *AVS* que ha sido retirado del canal no se almacena en $s_z.set\_st\_avs_x$ (antecedente falso).

  El mensaje *AVS* que permite señalar al nodo x como *victim* verifica que

$first(m.path).id = last(m.path).id = x$, es decir, $n_1 = x \wedge n_i = x$ porque $penult(m.path) \in s_{z-1}.setPred_x$. Por lo tanto, la ruta que contiene el mensaje representa un ciclo real.

Para otras rutas en las que esté presente el nodo x:

- $x = n_1\prime$:
  1. ruta contenida en $s_{z-1}.set\_st\_avs_{n_i\prime}$. Todos los nodos hasta $n_i\prime$ coinciden con los del ciclo.
     Cuando $n_{i\prime} \neq x$, la propiedad que se está demostrando establece que la existencia del mensaje *AVS* que señala víctima implica que no existe ningún otro mensaje almacenado o dirigido a cualquier otro nodo, en cuya ruta el primer nodo es precisamente $(x,\ t_x)$.
  2. ruta contenida en un mensaje *AVS* hacia $n_j\prime$. Ya sea con $n_i\prime = n_j\prime$ o $n_i\prime \neq n_j\prime$, la hipótesis inductiva indica que no puede haber ningún mensaje, a excepción del que señala víctima, que tenga como primer elemento de su ruta el par $(x,\ t_x)$.
  3. ruta contenida en $s_{z-1}.st\_avsrsp_x$ o en un mensaje *AVSRSP* con destino el nodo x.
     a) Caso $n_i\prime = n_j\prime$: Aplicando la propiedad B.2.3 es fácilmente comprobable que la ruta de $s_{z-1}.st\_avsrsp_x$ está contenida en el mensaje *AVS* hasta $n_i\prime$. Como el mensaje que señala víctima al nodo x cumple la propiedad, la hipótesis inductiva indica que $s_{z-1}.status.alg_{n_i\prime} \in \{dummy,\ candidate\}$ porque $n_i\prime = n_l$. En caso de ser *candidate*, además se cumple $s_{z-1}.inf\_need_{n_i\prime} = true \wedge sid_{AVS} = s_{z-1}.sim\_id_x > s_{z-1}.sim\_id_{n_l}$.
     La propiedad considera que, cuando $s_{z-1}.status.alg_{n_i\prime} = candidate$, $s_{z-1}.sim\_id_{n_i\prime} \geq sid_{AVSRSP}$. Como $sid_{AVSRSP} > s_{z-1}.sim\_id_x$, finalmente se obtiene que $s_{z-1}.sim\_id_{n_i\prime} > s_{z-1}.sim\_id_x$. Por otra parte, el mensaje *AVS* verifica que $s_{z-1}.sim\_id_{n_i\prime} < s_{z-1}.sim\_id_x$. Como las relaciones son opuestas, se concluye que es imposible que haya un mensaje almacenado en $s_{z-1}.st\_avsrsp_x$ o un mensaje *AVSRSP* hacia el nodo x con $x = n_1\prime$ si $n_i\prime = n_j\prime$.
     b) Caso $n_i\prime \neq n_j\prime \wedge n_i\prime = n_1\prime$: El antecedente de la propiedad establece que $(n_1\prime,\ t_1\prime) \notin s_{z-1}.setPred_{n_2\prime} \wedge s_{z-1}.blocker_{n_1\prime} = n_2\prime$. Como $x = n_1$ en la ruta del mensaje *AVS*, este nodo forma parte del camino *wait* y, por tanto, $(n_1,\ t_1) \in s_{z-1}.setPred_{n_2} \wedge s_{z-1}.blocker_{n_1} = n_2)$. Resulta evidente que $n_2 = n_2\prime$, mostrando así la imposibilidad de esta situación.
     c) Caso $n_i\prime \neq n_j\prime \wedge n_i\prime \neq n_1\prime$: En esta situación, la propiedad establece que, o bien $s_{z-1}.status.alg_{n_i\prime} = candidate \wedge s_{z-1}.inf\_need_{n_i\prime}$

$= true$, o $s_{z-1}.status.alg_{n_i\prime} = dummy$, siendo $s_{z-1}.blocker_{n_i\prime} = n_{i\prime+1}$ en ambos casos. En el mensaje AVS $n_i\prime$ forma parte de un camino *wait* porque $n_l = n_{i\prime+1}$. Esto implica que $s_{z-1}.blocker_{n_i\prime} = k \wedge k = n_i\prime$. Como el nodo k también pertenece al camino *wait*, $(n_{i\prime}, t_{i\prime}) \in s_{z-1}.setPred_k$, pero el antecedente de la propiedad exige que $(n_{i\prime}, t_{i\prime}) \notin s_{z-1}.setPred_{n_{i\prime+1}}$. Así que, se llega a la conclusión de que no existe un mensaje almacenado en $s_{z-1}.st\_avsrsp_x$ o un mensaje $AVSRSP$ hacia el nodo x con $x = n_1\prime$ si $n_i\prime \neq n_j\prime$.

- $x = n_l\prime$: En este supuesto es evidente que la ruta del mensaje AVS coincide con las otras rutas que se estudian desde el nodo x hasta el nodo $n_i\prime$.
  1. la ruta está contenida en $s_{z-1}.set\_st\_avs_{n_i\prime}$. Aplicando la propiedad B.2.36 al nodo x de la ruta almacenada en $s_{z-1}.set\_st\_avs_{n_i\prime}$, se determina que $s_{z-1}.st\_avsrsp_x \neq NULL$ ya que el nodo es *candidate*, $s_{z-1}.inf\_need_x = true$ y, por supuesto, $s_{z-1}.status.id_x = known$. Sin embargo, el nodo x, que pasará a ser víctima, ocupa la primera posición de la ruta del mensaje AVS y la propiedad indica que $s_{z-1}.st\_avsrsp_x = NULL$. Esto quiere decir que no es posible que a la vez que un mensaje AVS contiene un ciclo, exista almacenado un mensaje en el que $x = n_l\prime$.
  2. la ruta está contenida en un mensaje AVS que se dirige a $n_j\prime$, ya sea $n_i\prime = n_j\prime \vee n_i\prime \neq n_j\prime$.
     La propiedad B.2.36 establece que el nodo x de la ruta del mensaje AVS con destino $n_j\prime$ posee una ruta almacenada en $s_{z-1}.st\_avsrsp_x$. Por el contrario, el nodo x que pasará a ser víctima ocupa la primera posición de la ruta del mensaje AVS y la propiedad asegura que $s_{z-1}.st\_avsrsp_x = NULL$. Esto quiere decir que no es posible que a la vez que un mensaje AVS que contiene un ciclo, exista otro mensaje AVS en el que $x = n_l\prime$.
  3. la ruta aparece contenida en $s_{z-1}.st\_avsrsp_{n_1\prime}$ o en un mensaje $AVSRSP$ con destino $n_1\prime$.
     Caso $n_i\prime = n_j\prime$: La propiedad asegura que $sid_{AVSRSP} > s_{z-1}.sim\_id_{n_1\prime}$ y $s_{z-1}.sim\_id_{n_i\prime} \geq sid_{AVSRSP}$. Además, por ser $x = n_l\prime$, se sabe que $sid_{AVSRSP} > s_{z-1}.sim\_id_x$. Considerando estas expresiones, se puede afirmar que $s_{z-1}.sim\_id_{n_i\prime} > s_{z-1}.sim\_id_x$.
     Sin embargo, por el hecho de que el nodo x también forme parte de la ruta del mensaje AVS que contiene un ciclo, se cumple que: $sid_{AVS} = s_{z-1}.sim\_id_x \wedge sid_{AVS} > s_{z-1}.sim\_id_{n_i\prime}$ porque $n_i\prime = n_l$ en el mensaje AVS que contiene el ciclo. Sustituyendo adecuadamente estas expresiones se llega a la conclusión de que

$s_{z-1}.sim\_id_x > s_{z-1}.sim\_id_{n_i\prime}$. La comparación de los requisitos que verifica $s_{z-1}.sim\_id_x$ pone de manifiesto que la situación que se analiza es imposible.
El caso $n_i\prime \neq n_j\prime \wedge n_i\prime = n_1\prime$ no se tiene en cuenta porque el nodo x no puede ocupar la posición $n_l\prime$ cuando $n_i\prime = n_1\prime$.
Caso $n_i\prime \neq n_j\prime \wedge n_i\prime \neq n_1\prime$:
De acuerdo con la posición que ocupa $n_{i\prime}$ en el mensaje *AVS* es evidente que $s_{z-1}.status.alg_{n_i\prime} = dummy \vee (s_{z-1}.status.alg_{n_i\prime} = candidate \wedge s_{z-1}.inf\_need_{n_i\prime} = true)$. En cualquiera de esos estados, es necesario que se cumpla que $s_{z-1}.blocker_{n_i\prime} = n_{i\prime+1}$.
Como el nodo $n_i\prime$ está incluido en la ruta del mensaje *AVS*, se sabe que forma parte un camino *wait* y, por tanto, $s_{z-1}.blocker_{n_i\prime} = k \wedge (n_{i\prime},\ t_{i\prime}) \in s_{z-1}.setPred_k$. De todo esto, se deduce que $n_{i\prime+1} = k$ pero el antecedente de la propiedad requiere que $(n_{i\prime},\ t_{i\prime}) \notin s_{z-1}.setPred_{n_{i\prime+1}}$. En conclusión, resulta imposible que haya un mensaje de tipo *AVS* que señale víctima al nodo x y al mismo tiempo este nodo esté contenido en un mensaje *AVSRSP* o almacenado en *st_avsrsp*, siendo $x = n_{l\prime}$.

- $x = n_i\prime$
  1. la ruta está contenida en $s_{z-1}.set\_st\_avs_x$. Al ejecutarse la acción $s_z.status.alg_{n_i\prime} = victim$ y la propiedad sigue cumpliéndose para esta ruta.
  2. la ruta está contenida en un mensaje *AVS* que se dirige a $n_{j\prime}$, ya sea $n_i\prime = n_j\prime \vee n_{i\prime} \neq n_{j\prime}$. Los efectos de la acción modifican $s_z.status.alg_{n_i\prime}$. Este cambio no afecta a la propiedad porque el nuevo estado está también incluido en la misma.
  3. la ruta aparece contenida en $s_{z-1}.st\_avsrsp_{n_1\prime}$ o en un mensaje *AVSRSP* con destino $n_1\prime$.
     El cambio de estado del nodo $n_i\prime$ es posible cuando $n_i\prime = n_j\prime$ ya que si $s_z.status.alg_{n_i\prime} = victim$ la propiedad se sigue cumpliendo.
     Cuando $n_i\prime \neq n_j\prime \wedge n_i\prime = n_1\prime$, se observa que la ruta analizada no puede existir. El antecedente de la propiedad establece que $(n_1\prime,\ t_1\prime) \notin s_{z-1}.setPred_{n_2\prime} \wedge s_{z-1}.blocker_{n_1\prime} = n_2\prime$. Como $x = n_1$ en la ruta del mensaje *AVS*, este nodo forma parte del camino *wait* y, por tanto, $(n_1,\ t_1) \in s_{z-1}.setPred_{n_2} \wedge s_{z-1}.blocker_{n_1} = n_2$. Es evidente que $n_2 = n_2\prime$, confirmando así que esta situación es imposible.
     Para $n_i\prime \neq n_j\prime \wedge n_i\prime \neq n_1\prime \wedge s_{z-1}.inf\_need_{n_i\prime} = false$ y $s_{z-1}.status.alg_{n_i\prime} = candidate$, la propiedad se sigue verificando. Si $s_{z-1}.inf\_need_{n_i\prime} = true \wedge s_{z-1}.blocker_{n_i\prime} = n_{i\prime+1}$, al ejecutar la acción $s_z.status.alg_{n_i\prime}$

$= victim \wedge s_z.inf\_need_{n_{i\prime}} = false$. Sin embargo, como el nodo x está incluido en la ruta del mensaje *AVS*, se sabe que forma parte un camino *wait* y, por tanto, $s_{z-1}.blocker_x = k \wedge (x,\ t_x) \in s_{z-1}.setPred_k$. De todo esto, se deduce que $n_{i\prime+1} = k$, pero el antecedente de la propiedad requiere que $(n_{i\prime},\ t_{i\prime}) \notin s_{z-1}.setPred_{n_{i\prime+1}}$. En conclusión, resulta imposible que haya un mensaje de tipo *AVS* que señale víctima al nodo x y al mismo tiempo este nodo esté contenido en un mensaje *AVSRSP* o almacenado en *st_avsrsp*, siendo $x = n_{i\prime}$.
Según este análisis, queda comprobado que la propiedad en s_z es cierta en esta situación.

- $x > n_{i\prime}$: La propiedad recoge esta posibilidad cuando $x \in ewait$. Para que la acción esté habilitada es preciso que $s_{z-1}.status.id_x = known$, pero los nodos que forman parte de un camino *ewait* son *unknown*.

Caso $x \in p\prime$ tal que $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$:

La definición de *ewait* obliga a que $s_{z-1}.status.id_x = unknown$, pero la condición de habilitación exige que $s_{z-1}.status.id_x = known$.

- $\pi_z \in \{rcvAVSRSP_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{AVSRSP}\}$. El único efecto de la acción que puede afectar al cumplimiento de la propiedad es el que consiste en almacenar íntegramente en $s_z.st\_avsrsp_x$ el mensaje *AVSRSP* que se retira del canal. Como el mensaje *AVSRSP* cumplía la propiedad en s_{z-1} y las variables implicadas no se modifican, la hipótesis inductiva concluye.

- $\pi_z \in \{dltINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. La precondición de esta acción exige la presencia de un mensaje *INF* en el canal para ejecutarse. A continuación, se van a analizar los casos en los que el consecuente es cierto en s_{z-1} debido a la existencia de un mensaje *INF* en el canal.

  Cuando el nodo x se corresponde con el nodo n_i y el nodo y es cualquiera, se cumple que el tiempo asociado al mensaje *INF* es $s_{z-1}.t\_unk_{n_i}$. La acción sólo podría estar habilitada si $s_{z-1}.t\_unk_{n_i}$ vale -1. Aplicando la propiedad B.2.2, se obtiene que $1 \leq s_{z-1}.t\_unk_{n_i} \leq s_{z-1}.ta_{n_i}$. Así que la acción no está habilitada. En el caso de que el mensaje *INF* vaya dirigido al último elemento de un camino *ewait* tal que $x = last(p\prime).id$, se requiere el tiempo incluido en el mensaje *INF* sea precisamente $s_{z-1}.t\_unk_{last(p\prime).id}$. Como la propiedad B.2.2 establece que $1 \leq s_{z_1}.t\_unk_{last(p\prime).id}$, la acción no se podrá ejecutar.

- $\pi_z \in \{rcvINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Esta acción puede llegar a cambiar las variables sim_id_x y $status.alg_x$. Una de las condiciones para que la acción esté habilitada es que $s_{z-1}.status.id_x = unknown$, pero los efectos de la

acción hacen que $s_z.status.id_x$ se convierta en *known*. Por la propiedad B.2.15, se deduce que entonces $s_{z-1}.status.alg_x \in \{active, blocked\}$. Suponiendo que la propiedad se cumple en s_{z-1} con antecedente y consecuente ciertos, el nodo x sólo puede hacer las veces del nodo n_i que señala la propiedad o de un nodo que ocupe posiciones posteriores al nodo n_i en una posible ruta.

A continuación, se analiza qué ocurre cuando se ejecuta la acción y en s_{z-1} existe una ruta p de las permitidas por la propiedad.

Caso $x = n_i$:

Ruta almacenada $s_{z-1}.set\_st\_avs_{n_i}$. Al recibirse el mensaje *INF*, se seguirá cumpliendo la propiedad porque $s_z.status.alg_{n_i} \in \{active, candidate\}$ y son estados permitido para el nodo n_i en s_z. El resto de características de la ruta no varía. Los mensajes almacenados en $s_{z-1}.set\_st\_avs_{n_i}$ se eliminan y reconvierten en mensajes *AVSRSP* siempre y cuando $s_z.sim\_id_{n_i}$ sea superior a su campo *sid*, $s_{z-1}.inf\_need_{n_i} = true$ y $m.sid > s_{z-1}.sim\_id_{n_i}$. La hipótesis inductiva permite suponer que el mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ cumple la propiedad. En consecuencia, hay que comprobar que el nuevo mensaje *AVSRSP* también lo hace:

1. $s_z.status.alg_{n_1} = candidate$. Sólo hay que tener en cuenta que el primer nodo de la ruta almacenada $s_{z-1}.set\_st\_avs_{n_i}$ coincide con el primer nodo de la ruta del mensaje *AVSRSP* formado y el estado de ese nodo no cambia en el efectos analizado.
2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i,\ t_i)$. Considerando la propiedad B.2.21 y la hipótesis inductiva se establece que el último elemento de la ruta almacenada en $s_{z-1}.set\_st\_avs_{n_i}$ es precisamente el nodo n_i. Por tanto, es evidente que el nodo n_i de la propiedad para el mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ se mantiene en el mensaje *AVSRSP* generado. Como las rutas son exactamente iguales y no se modifican las variables *ta* y *setPred* de ningún nodo, esta característica se verifica.
3. $\forall\ l < i$, $s_z.status.id_{n_l} = known$. La hipótesis inductiva asegura que este requisito se cumple en s_{z-1} para todos los nodos de la ruta almacenada en $s_{z-1}.set\_st\_avs_{n_i}$. Como la ruta del mensaje *AVSRSP* es la misma y no se modifica la variable *status.id* de ningún nodo, la condición se cumple.
4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Este punto también se verifica porque las rutas consideradas son iguales. Además, los efectos de la acción no provocan ningún cambio de estado.
5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Aplicando la hipótesis inductiva, se llega a la

conclusión de que los nodos de la ruta del mensaje *AVSRSP* verifican el requisito.

Al ejecutarse la acción y generarse el mensaje *AVSRSP* también se verifica que:

- El campo *sid* del mensaje *AVSRSP* contiene un valor superior a $s_z.sim\_id_{n_1}$. Se llega a esa conclusión porque el campo *sid* del mensaje *AVSRSP* adopta como valor $s_z.sim\_id_{n_i}$ y para que se produzca este efecto se tiene que cumplir que $s_{z-1}.sim\_id_{n_i}$ sea mayor que el campo *sid* del mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$. Además, la creación del mensaje *AVSRSP* va acompañado de otro efecto por el que el nodo n_i incluye el valor de *m.sid* como su identidad simulada. De esta forma, $s_z.sim\_id_{n_i} > s_{z-1}.sim\_id_{n_i}$ y el campo *sid* del mensaje *AVSRSP* es superior tanto al campo *sid* del mensaje almacenado $s_{z-1}.set\_st\_avs_{n_i}$ como a $s_z.sim\_id_{n_1}$ porque coincide con $s_{z-1}.sim\_id_{n_1}$ y la identidad simulada del nodo n_1 no cambia.
- Tras la ejecución de la acción y formarse el mensaje *AVSRSP* se sabe que: $s_z.status.alg_{n_i} \in \{active, candidate\}$, $s_z.status.id_{n_i} = known$ y $s_z.inf\_need_{n_i} = false$. Dado que el nodo n_i coincide con el nodo n_j (propiedades B.2.16 y B.2.21) basta con comprobar que en esta situación que $s_z.sim\_id_{n_i} \geq sid$. Es obvio que esta condición se cumple porque al formarse el mensaje *AVSRSP* se impone que $s_z.sim\_id_{n_i} = sid$.

Si el mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ se mantiene al ejecutar la acción, basta con asegurar que $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_i}$ para afirmar que la propiedad se cumple. La hipótesis inductiva establece que $s_{z-1}.sim\_id_{n_1} > s_{z-1}.sim\_id_{n_i}$ y $sid = s_{z-1}.sim\_id_{n_1}$. Al ejecutarse la acción en este supuesto, tanto el campo *sid* del mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ como la variable *sim_id* del nodo n_1 no cambian de valor. Sin embargo, el valor de $s_z.sim\_id_{n_i}$ puede varíar dependiendo del valor de $s_{z-1}.inf\_need_{n_i}$.

- Si $s_{z-1}.inf\_need_{n_i} = false$, se debe estudiar si es posible alcanzar esta situación. Como $n_i = n_j$, es evidente que el último nodo de la ruta es n_i. Esto implica que el nodo n_i se incorporó a la ruta ejecutando las acciones que permiten que ocupe la última posición de la ruta. El estado del nodo n_i en esos casos es *candidate*. Por otra parte, la precondición de la acción indica que $s_{z-1}.status.id_{n_i} = unknown$. La única acción que permite alcanzar este estado es la acción $EndDelArc_{n_i}(n)$. Además, la propiedad B.2.50 asegura que el nodo n_i, que era *candidate* al iniciarse la formación de la ruta, nunca llega a alcanzar el estado *dummy* si mantiene inalterado su correspondiente conjunto *setPred*. En consecuencia, antes de la ejecución de la acción $EndDelArc_{n_i}(n)$, el estado del nodo n_i sólo puede ser *candidate*. Según la propiedad B.2.50, podría pensarse que el estado del nodo

n_i antes de ejecutar $EndDelArc_{n_i}(n)$ pudiera ser *blocked* y *unknown*. Dado que este estado precisa a su vez la ejecución de la misma acción para que $status.id_{n_i}$ se convierta en *unknown*, la situación previa coincide con la que se pretende estudiar en este punto. Por todo ello, las dos posibilidades que hay que analizar son:

1. $status.alg_{n_i} = candidate$ e $inf\_need_{n_i} = false$. Si el nodo n_i tiene que recibir el mensaje *INF* es necesario que, previamente, se haya recibido un mensaje *ALG* que haga que $s_{z-1}.t\_unk_{n_i}$ sea distinto de -1. Para que este mensaje exista, el nodo n_i ha tenido que bloquearse de nuevo porque los efectos de la acción hicieron que $status.alg_{n_i} = active$. Se sabe que la acción $StartAddArc_{n_i}(n\prime)$ modifica el estado del nodo n_i y pasa a ser *candidate* porque $setPredToInf_{n_i} \neq \emptyset$. Se comprueba que $setPredToInf_{n_i} \neq \emptyset$, al observar que $(n_{i-1}, t_{i-1}) \in setPred_{n_i}$. De acuerdo a la propiedad B.2.47, se deduce que $(n_{i-1}, t_{i-1}) \in setPredToInf_{n_i}$. Como se vuelve al estado inicial del nodo n_i para este supuesto, queda demostrado que no se dan las condiciones para la recepción del mensaje *INF* analizado.
2. $status.alg_{n_i} = candidate$ e $inf\_need_{n_i} = true$. Al ejecutarse $EndDelArc_{n_i}(n)$, $status.alg_{n_i} = active$ y $status.id_{n_i} = known$. Como el caso que se estudia precisa que $s_{z-1}.inf\_need_{n_i} = false$, es obvio que se debe recibir un mensaje *INF* para que cambie su valor. Las acciones que convierten el estado del nodo n_i en *victim* también transforman $inf\_need_{n_i}$ en *false*. Sin embargo, en ese estado el nodo x podría llegar a tener habilitada solamente la acción $Abort_{n_i}$ y, en consecuencia, el estado del nodo pasaría a ser *aborted*. Alcanzado ese estado, el nodo no puede cambiar de estado de nuevo. La única acción que permite recibir un mensaje *INF* hace que el estado del nodo n_i también se vea modificado y vuelva a ser *candidate* o pasar a ser *active* y *known* ya que $setPredToInf_{n_i} \neq \emptyset$. Siendo *candidate* el nodo n_i, resulta evidente que no se consigue de ninguna manera cumplir la precondición de la acción en la que $s_{z-1}.status.id_{n_i} = unknown$. Si $status.alg_{n_i} = active$ y $status.id_{n_i} = known$, el nodo n_i tendrá que bloquearse de nuevo y recibir un mensaje *ALG* para que $s_{z-1}.t\_unk_{n_i} \neq -1$ y la propiedad B.2.2 señala que $1 \leq m.ta \leq s_{z-1}.ta_{n_i}$. No hay que olvidar que la recepción del mensaje *INF*, además de convertir *inf_need* a *false*, hace que $t\_unk_{n_i} = -1$. Al bloquearse el nodo n_i, $status.alg_{n_i} = candidate$. En consecuencia, se impide que el nodo alcance las precondiciones de la acción $rcvINF_{n_i}(y, m)$, siendo $s_{z-1}.inf\_need_{n_i} = false$.

- Si $s_{z-1}.inf\_need_{n_i} = true$, $s_z.sim\_id_{n_i}$ puede conservar el valor que tenía $s_{z-1}.sim\_id_{n_i}$ o tomar el valor de $m_{INF}.sid$ cuando $m.sid > s_{z-1}.sim\_id_{n_i}$.

En el primer caso, la hipótesis inductiva concluye. En el segundo caso hay que tener en cuenta que el campo *sid* del mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ y $s_{z-1}.sim\_id_{n_1}$ no se modifican. Tras la ejecución de la acción resulta que $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_i}$. La condición que impide que se forme un mensaje *AVSRSP* y, por tanto, el mensaje de $s_{z-1}.set\_st\_avs_{n_i}$ siga almacenado en s_z es $s_z.sim\_id_{n_i} \leq sid$. La propiedad B.2.57 descarta la igualdad en esta condición. Así que, $s_z.sim\_id_{n_i} < sid$ como señala la propiedad.

Ruta de un mensaje AVS dirigido al nodo n_j. Este mensaje se mantiene en el canal en s_z cumpliendo todos los requisitos impuestos por la propiedad. Como $s_z.status.alg_{n_i} \in \{active, candidate\}$, se observa que, tanto si $n_i = n_j$ como si $n_i \neq n_j$, el estado del nodo n_i verifica la propiedad.

Ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ *o incluida en un mensaje AVSRSP que va hacia el nodo* n_1. Para confirmar que este tipo de rutas verifican la propiedad se analizan todas las posibles situaciones en las que la acción está habilitada:

- Suponiendo que $n_i = n_j$ y $s_{z-1}.inf\_need_{n_i} = false$, se sabe que el estado final del nodo n_i es uno de los permitidos por la propiedad ya que $s_z.status.alg_{n_i} \in \{active, candidate\}$. Como $n_i = n_j$ es evidente que $last(p).id = n_i$. Este nodo se incorporó a la ruta ejecutando las acciones que permiten que el nodo n_i ocupe la última posición de la ruta p, es decir, que aparezca como único elemento de la ruta. El estado del nodo n_i en todo esos casos pasa a ser *candidate* y *known*.

  La precondición de la acción establece que $s_{z-1}.status.id_{n_i} = unknown$. La única acción que permite alcanzar este estado es la acción $EndDelArc_{n_i}(n)$. Por otro lado, la propiedad B.2.50 asegura que el nodo n_i, que era *candidate* cuando empezó a formarse la ruta p, no llega a ser *dummy* si mantiene inalterado su correspondiente $setPred_{n_i}$. El único estado posible que el nodo n_i puede presentar antes de la ejecución de la acción $EndDelArc_{n_i}(n)$ es, según la propiedad B.2.50, *candidate*. Podría pensarse que el estado del nodo n_i antes de ejecutar $EndDelArc_{n_i}(n)$ sea *blocked* y *unknown*. Este estado precisa de la ejecución de la acción $EndDelArc_{n_i}(n)$ para que $status.id_{n_i} = unknown$. Así que, la situación previa coincide con la que se pretende estudiar en este apartado y, antes de ejecutar $EndDelArc_{n_i}(n)$, el estudio se reduce a los siguientes casos:

  1. $status.alg_{n_i} = candidate$ e $inf\_need_{n_i} = false$. La variable $t\_unk_{n_i}$ pasa

a valer -1. Si el nodo n_i tiene que recibir el mensaje *INF* considerado $m.ta = s_{z-1}.t\_unk_{n_i}$, es necesario que previamente haya recibido un mensaje *ALG* que haga que $s_{z-1}.t\_unk_{n_i} \neq -1$. Para que este mensaje exista, el nodo n_i tuvo que bloquearse de nuevo porque los efectos de la acción hicieron que $status.alg_{n_i} = active$. La acción $StartAddArc_{n_i}(n')$ modifica el estado del nodo n_i pasando a ser *candidate* porque $setPredToInf_{n_i} \neq \emptyset$. Se comprueba que $setPredToInf_{n_i} \neq \emptyset$, al observar que $(n_{i-1},\ t_{i-1}) \in setPred_{n_i}$. De acuerdo a la propiedad B.2.47, se deduce entonces que $(n_{i-1},\ t_{i-1}) \in setPredToInf_{n_i}$. Como se vuelve al estado inicial del nodo n_i para este supuesto, queda demostrado que no se dan las condiciones para la recepción del mensaje *INF* en las condiciones consideradas.

2. $status.alg_{n_i} = candidate$ e $inf\_need_{n_i} = true$. Al ejecutar $EndDelArc_{n_i}(n)$, $status.alg_{n_i} = active$ y $status.id_{n_i} = known$. Como el caso que se estudia precisa que $s_{z-1}.inf\_need_{n_i} = false$, es obvio que se debe recibir un mensaje *INF* de tal forma que $inf\_need_{n_i}$ cambie su valor. Nótese que las acciones que convierten el estado del nodo n_i en *victim* transforman $inf\_need_{n_i} = false$. Sin embargo, en ese estado el nodo n_i sólo podría tener habilitada la acción $Abort_{n_i}$ y, en consecuencia, el nodo pasaría a ser *aborted*. Alcanzado ese estado, el nodo n_i no podrá cambiar de estado de nuevo. La acción que permite recibir un mensaje *INF* cambia también el estado del nodo n_i y vuelve a ser *candidate* o pasa a ser *active* y *known* ya que $setPredToInf_{n_i} \neq \emptyset$. Siendo *candidate* el estado del nodo n_i, resulta evidente que no se consigue de ninguna manera cumplir la precondición de la acción, $s_{z-1}.status.id_{n_i} = unknown$. Por otra parte, si $status.alg_{n_i} = active$ y $status.id_{n_i} = known$, el nodo n_i tendrá que bloquearse de nuevo y recibir un mensaje *ALG* para que $s_{z-1}.t\_unk_{n_i} \neq -1$ porque $m.ta = s_{z-1}.t\_unk_{n_i}$ y la propiedad B.2.2 señala que $1 \leq m.ta \leq s_{z-1}.ta_{n_i}$. No hay que olvidar que la recepción del mensaje *INF*, además de convertir $inf\_need_{n_i} = false$, tiene como efecto que $t\_unk_{n_i}$ pase a valer -1. Al bloquearse el nodo n_i, $status.alg_{n_i} = candidate$. En consecuencia, se impide que el nodo alcance las precondiciones de la acción $rcvINF_{n_i}(y,\ m)$.

- Supóngase ahora que $n_i = n_j$ y $s_{z-1}.inf\_need_{n_i} = true$. Es obvio que entonces la propiedad se cumple cuando $s_z.sim\_id_{n_i}$ toma el valor de $s_{z-1}.sim\_id_{n_i}$. En el caso de que $s_z.sim\_id_{n_i}$ adopte el valor del campo *sid* del mensaje *INF* recibido, se sabe que entonces se cumplía que $m.sid > s_{z-1}.sim\_id_{n_i}$. Sustituyendo esta condición, se obtiene que $s_z.sim\_id_{n_i} > sid$ y se verifica la propiedad.

- Si $n_i \neq n_j$, $n_i \neq n_1$ y $s_{z-1}.inf\_need_{n_i} = true$, al ejecutarse la acción, los efectos modifican las siguientes variables asociadas al nodo n_i: $s_z.status.alg_{n_i} \in \{active, candidate\}$, $s_z.status.id_{n_i} = known$ y $s_z.inf\_need_{n_i} = false$. En esta situación, la propiedad indica que $sid \leq s_z.sim\_id_{n_i}$ y es necesario comprobar que esta condición se cumple tras retirar el mensaje *INF* del canal. La ejecución de la acción puede conservar el valor de $sim\_id_{n_i}$ o adoptar el campo *sid* del mensaje *INF* siempre que $s_{z-1}.sim\_id_{n_i} < m.sid$. La hipótesis inductiva establece dos posibilidades distintas:
  1. $(sid, p - first(p)) = s_z.st\_alg_{n_1}$. El mensaje *INF* directo que se retira del canal como efecto de la acción cumple que $sid \leq s_{z-1}.sim\_id_{n_i}$. Sea cual sea el valor que adopta $s_z.sim\_id_{n_i}$ al ejecutarse la acción, si se sustituye en la expresión anterior, resulta que $sid < s_z.sim\_id_{n_i}$ tal y como señala la propiedad en las circunstancias resultantes en s_z.
  2. $(sid, p - first(p)) \neq s_z.st\_alg_{n_1}$. En estas condiciones, el mensaje *INF* directo que llega al nodo n_i cumple que $m.sid > sid$ y $sid > s_{z-1}.sim\_id_{n_i}$. Estas dos relaciones equivalen a $m.sid > s_{z-1}.sim\_id_{n_i}$. Considerando la situación resultante en s_z, la propiedad exige que $sid \leq s_z.sim\_id_{n_i}$. Para comprobar esto, basta con incluir la relación existente entre $s_z.sim\_id_{n_i}$ y $m.sid$. Siendo $m.sid > s_{z-1}.sim\_id_{n_i}$, se asume que $s_z.sim\_id_{n_i} = m.sid$. De este modo, se llega a la conclusión de que $sid \leq s_z.sim\_id_{n_i}$.
- Si $n_i \neq n_j$, $n_i \neq n_1$ y $s_{z-1}.inf\_need_{n_i} = false$, la acción tiene como efecto que $s_z.sim\_id_{n_i} = m.sid$. Además, la hipótesis inductiva asegura que en s_{z-1}, $m.sid > sid$ ya sea cuando $s_{z-1}.st\_alg_{n_1}$ coincida o no con $(sid, p - first(p))$. Esto implica que, tras la ejecución de la acción, se cumple que $n_i \neq n_j$ y $s_z.sim\_id_{n_i} > sid$ tal y como indica la propiedad cuando $s_z.status.alg_{n_i} \in \{active, candidate\}$, $s_z.status.id_{n_i} = known$ y $s_z.inf\_need_{n_i} = false$.

Caso $x \in p\prime$ tal que $ewait(n_i, last(p\prime).id, p\prime, s_{z-1})$:

De todas las rutas mencionadas en la propiedad, se deduce que no es posible disponer de una ruta almacenada en $s_{z-1}.set\_st\_avs_x$ en la que el último nodo es n_i y que, al mismo tiempo, el nodo x ocupa posiciones posteriores al nodo n_i. Si se considera la ruta de un mensaje *AVS* hacia el nodo n_j, sólo si $n_i \neq n_j$ se podría estar en el caso analizado. En esa situación, los efectos de la acción no modifican ni las variables de la propiedad asociadas al nodo n_i ni las del nodo n_1 y, por tanto, la propiedad es cierta en s_z. El estudio de los efectos de la acción que afectan a una ruta almacenada en $s_{z-1}.st\_avsrsp_{n_1}$ o de un mensaje *AVSRSP* hacia el nodo n_1, sólo es posible cuando $n_i \neq n_j$ y el nodo x está presente en el camino *ewait*. La existencia de un mensaje *INF* que es recibido por el nodo x permite asumir que que $x = last(p\prime).id$. Así que los efectos de la acción hacen

que se genere un nuevo mensaje *INF* directo o a través de un camino *ewait*. Según los diferentes valores que pueden adoptar las variables correspondientes a n_i en s_{z-1} se tienen diversos casos:

- $s_{z-1}.inf\_need_{n_i} = true$ y $(sid,\ p - first(p)) = s_{z-1}.st\_alg_{n_1}$. Mientras existan nodos en el camino *ewait*, se generará un mensaje *INF* dirigido al último nodo de $p\prime$. De acuerdo con la relación de orden, el campo *sid* de este nuevo mensaje será mayor que $s_z.sim\_id_x$. Se comprueba fácilmente que los efectos de esta acción no modifican ni $sim\_id_{n_i}$ ni el correspondiente *sid* de los mensajes analizados. Eso quiere decir que, tras la ejecución de la acción, la relación $sid \leq s_z.sim\_id_{n_i}$ se sigue verificando, cualquiera que sea el valor de $s_{z-1}.sim\_id_x$. Cuando el destino del mensaje *INF* coincide con el nodo n_i, el camino *ewait* se ha extinguido y el mensaje *INF* se considera ya directo.
- $s_{z-1}.inf\_need_{n_i} = true$ y $(sid,\ p - first(p)) \neq s_{z-1}.st\_alg_{n_1}$. En esta situación, $s_z.sim\_id_x$ puede tomar dos valores distintos:
  1. $s_z.sim\_id_x = s_{z-1}.sim\_id_x$. En s_{z-1} se cumple, por hipótesis inductiva, que $m.sid > sid$. El campo *sid* del nuevo mensaje *INF* verifica que es superior a $s_z.sim\_id_x$ y, por lo tanto, mayor que *sid*. Esta relación es válida para todo mensaje *INF* que surge mientras exista el camino *ewait*. Del mismo modo, la hipótesis inductiva permite demostrar que $sid > s_z.sim\_id_{n_i}$. Siendo $sid > s_{z-1}.sim\_id_{n_i}$, tras la ejecución de la acción se verifica que $s_{z-1}.sim\_id_{n_i} = s_z.sim\_id_{n_i}$. Al eliminarse el camino *ewait*, el mensaje *INF* que va dirigido al nodo n_i también cumple las condiciones indicadas.
  2. $s_z.sim\_id_x = m.sid$. En el nuevo mensaje *INF*, el campo *sid* es superior a $s_z.sim\_id_x$. Como en s_{z-1} se verificaba la propiedad, $m.sid > sid$, se obtiene que el campo *sid* del nuevo mensaje *INF* es superior a *sid*. Cuando el mensaje *INF* generado tiene como destino el nodo n_i, la propiedad también se cumple. Además, mientras el mensaje *INF* recorre nodo a nodo el camino *ewait* hasta alcanzar el nodo n_i, la condición $sid > s_{z-1}.sim\_id_{n_i}$ se mantiene inalterada porque y $sim\_id_{n_i}$ no varía.
- $s_{z-1}.inf\_need_{n_i} = false$ y $(sid,\ p - first(p)) = s_{z-1}.st\_alg_{n_1}$. En la ejecución de la acción el nodo n_i y sus variables se mantienen sin cambios. El mensaje *INF* que se genera como efecto de la misma cumple la propiedad. Cuando se forma el mensaje *INF*, el camino $ewait(n_i,\ last(p\prime).id = x,\ p\prime,\ s_{z-1})$ se reduce hasta alcanzar el nodo n_i. En esa situación, el mensaje *INF* pasa a ser un mensaje directo con destino el nodo n_i en el que su campo *sid* es

mayor que el campo *sid* de los mensajes estudiados (ya comprobado en el caso $x = n_i$). Además, se deben considerar supuestos diferentes:

1. Si $last(p).id \in p\prime$, el mensaje *INF* se genera manteniendo los requisitos señalados en la propiedad hasta que el nodo n_j coincide con el último nodo de la ruta $p\prime$. Cuando $x = n_j$, el mensaje *INF* formado cumple que $s_{z-1}.sim\_id_{n_j} \geq sid$ y $s_{z-1}.inf\_need_{n_j} = true$. Al formarse un nuevo mensaje *INF*, siendo $s_{z-1}.inf\_need_{n_j} = true$, $s_z.sim\_id_{n_j}$ puede o bien mantener el valor de $s_{z-1}.sim\_id_{n_j}$ o bien adquirir el valor de $m.sid$ del mensaje *INF* retirado del canal. Las condiciones que cumple el mensaje *INF* que se envía al nodo n_j se corresponden con las que indica la propiedad para este caso que sigue ($last(p).id \notin p\prime$).
2. Si $last(p).id \notin p\prime$, como el nodo x debe pertenecer a la ruta $p\prime$ en s_{z-1}, es evidente que $s_{z-1}.inf\_need_x = false$ (hipótesis inductiva). Los efectos de la acción en estas condiciones consisten en que $s_z.sim\_id_x = m.sid$. Por otro lado, aplicando la relación de orden se obtiene que el campo *sid* del nuevo mensaje *INF* es mayor que $s_z.sim\_id_x$. Finalmente, todo ello, permite llegar a la conclusión de que el campo *sid* del nuevo mensaje *INF* tambiés es mayor que el campo *sid* de los mensajes analizados. Además, todos los nodos de la ruta $p\prime$, salvo el nodo x, en s_z, tienen a *false* la variable *inf_need*. Cuando el mensaje *INF* se envía al nodo n_i, se sigue cumpliendo que su campo *sid* es superior al campo *sid* de los mensajes estudiados, aunque el camino *ewait* se haya consumido.

- $s_{z-1}.inf\_need_{n_i} = false$ y $(sid,\ p - first(p)) \neq s_{z-1}.st\_alg_{n_1}$. La hipótesis inductiva indica que $m.sid > sid$ y el campo *sid* del nuevo mensaje *INF* que se forma como efecto de la acción se caracteriza porque es mayor que $s_z.sim\_id_x$ (relación de orden). En el caso de que $s_z.sim\_id_x$ adopte como valor $m.sid$, el campo *sid* del mensaje *INF* que surge a la vez que se reduce el camino *ewait* cumple que es superior a *sid*. Cuando $s_{z-1}.inf\_need_x = true$ y $m.sid \leq s_{z-1}.sim\_id_{n_i}$, sim_id_x conserva su valor en s_z. Dado que el campo *sid* del nuevo mensaje *INF* es superior a $s_z.sim\_id_x$, se deduce que también es mayor que el campo *sid* de los mensajes analizados. Así que, la ejecución repetida de la acción por parte de cada uno de los nodos del camino *ewait* considerado, mantiene la condición. Finalmente, cuando el mensaje *INF* se dirige al nodo n_i el requisito sigue cumpliéndose.

Un efecto de la acción puede modificar sim_id_x de manera que adopte el valor (x, $s_z.ta_x$, ϵ). Se puede observar que este efecto sucede cuando $s_z.setPredToInf_x = \emptyset$. En s_{z-1} el antecedente de la propiedad señala que $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_x$. Por tanto, de acuerdo con la propiedad B.2.47, $s_{z-1}.setPredToInf_x \neq \emptyset$. Como el

conjunto $setPred_x$ no cambia al ejecutar la acción, resulta que $s_z.setPredToInf_x \neq \emptyset$ y, en consecuencia, el efecto considerado no se produce.

- $\pi_z \in \{sndAVS_x\colon x \in \mathcal{N}\}$. Al ejecutarse la acción, se elimina un mensaje de $s_{z-1}.set\_st\_avs_x$ que podría llegar a hacer falso el antecedente de la propiedad. Pero, al mismo tiempo, se genera un mensaje de tipo *AVS* que debe verificar la propiedad. A continuación, se comprueban todas los requisitos que debe cumplir.

  1. $s_z.status.alg_{n_1} = candidate$. El mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ que se utiliza para formar la ruta del nuevo mensaje *AVS*, por hipótesis inductiva, cumple la propiedad y, por tanto, el estado del nodo que aparece en la primera posición de su ruta es *candidate*. Como el primer elemento de la ruta del mensaje *AVS* coincide con el de la ruta del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ y los efectos de la acción no modifican la variable *status.alg*, queda demostrado que $s_z.status.alg_{first(p).id} = candidate$.
  2. $p = \gamma_1{\cdot}\gamma_2$ con $last(\gamma_1) = (n_i,\, t_i)$. La ruta del nuevo mensaje *AVS* se forma superponiendo la ruta de un mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ con $s_{z-1}.st\_avsrsp_x.path$ a través del nodo x que tienen en común. Esto se deduce a partir de la propiedad B.2.21 que señala que el último elemento de un mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ coincide con el par $(x,\, t_x)$ y de la propiedad B.2.19 que afirma que $first(s_{z-1}.st\_avsrps_x.path) = (x,\, s_{z-1}.ta_x)$. Por otra parte, la propiedad B.2.16 asegura que el penúltimo elemento de un mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ está contenido en $s_{z-1}.setPred_x$. Como ésta es la condición que marca el antecedente de la propiedad para el nodo n_i, se concluye que el nodo x es precisamente el nodo n_i. La hipótesis inductiva también permite asegurar que los nodos de $s_{z-1}.st\_avsrps_x.path$ y del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ cumplen la condición analizada. Por todo ello, se concluye que la ruta del nuevo mensaje *AVS* verifica esta característica.
  3. $\forall\ l < i$, $s_z.status.id_{n_l} = known$. Aplicando por separado la hipótesis inductiva a las rutas que componen el mensaje *AVS* y considerando que la ruta de este nuevo mensaje se construye a partir de ellos, se demuestra el cumplimiento de este requisito.
  4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. A pesar de que la hipótesis inductiva establece que las rutas de $s_{z-1}.set\_st\_avs_x$ y $s_{z-1}.st\_avsrps_x.path$ cumplen este requisito, debe comprobarse que la unión de ambas rutas por medio del nodo x cumple la condición exigida a un nodo que es *candidate*. Como el campo *sid* del mensaje *AVS* es el que aparece en el mensaje almacenado en $s_{z-1}.set\_st\_avs_x$, es evidente que los nodos *candidate* que

pertenecen a esa ruta siguen cumpliendo la condición impuesta, o sea, *sid* $> s_z.sim\_id_{n_l}$. Por otra parte, al estar habilitada la acción, el valor de $s_{z-1}.cand\_succ_x$ es inferior o igual al campo *sid* de $s_{z-1}.set\_st\_avs_x$. Esta condición, según la propiedad B.2.20, se convierte en $s_{z-1}.st\_avsrps_x.sid$ es inferior o igual al campo *sid* de $s_{z-1}.set\_st\_avs_x$. De acuerdo con estas relaciones, el campo *sid* que adopta el mensaje *AVS* es superior a $s_z.sim\_id_{n_l}$, siendo n_l cualquiera de los nodos *candidate* que están presentes en $s_{z-1}.st\_avsrps_x.path$. Para el nodo x, común a las dos rutas involucradas, la hipótesis inductiva asegura que el campo *sid* del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ es $s_{z-1}.sim\_id_{n_1}$ y $s_{z-1}.sim\_id_{n_1} > s_{z-1}.sim\_id_x$. Al ejecutarse la acción no cambia sim_id_x y, por tanto, resulta que $sid > s_z.sim\_id_x$, tal y como se quería demostrar. Los efectos de la acción garantizan que $s_z.inf\_need_x = true$ y $s_z.status.alg_x = candidate$.

5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Tal y como se indica en el requisito anterior, los nodos a excepción del que ocupa la primera posición de las rutas son *dummy* o *candidate*. Considerando la propiedad B.2.29 se confirma que en esos estados la condición temporal se cumple. En el caso de los nodos de estado *candidate* además se tiene que aplicar la propiedad B.2.35 porque es necesario asegurar que cuando $s_z.inf\_need_{n_l} = true$ resulta que $s_z.st\_alg_{n_l} \neq NULL$. La hipótesis inductiva permite determinar que esta condición temporal también se cumple para el primer nodo de la ruta del mensaje *AVS* ya que éste coincide con el primer nodo de la ruta de $s_{z-1}.set\_st\_avs_x$.

Además de los requisitos anteriores, el nuevo mensaje *AVS* debe cumplir otras características:

- Si el nodo n_i coincide con n_j, nodo destino del mensaje *AVS*, la propiedad indica que sólo son posibles ciertos estados para el nodo n_i. Considerando la construcción de la ruta del mensaje *AVS* se sabe que el nodo n_i del mensaje *AVS* que señala la propiedad se corresponde con el nodo que verifica la propiedad para $s_{z-1}.st\_avsrps_x.path$. Como la propiedad se cumple en s_{z-1} para la ruta almacenada en $s_{z-1}.st\_avsrps_x$ y los efectos de la acción no cambian el valor las variables *status.alg* y *status.id* de ningún nodo, el estado del nodo n_i verifica que, según la hipótesis inductiva, cuando $n_i = n_j$ para $s_{z-1}.st\_avsrps_x$, se tienen los siguientes posibles estados: $s_z.status.id_{n_i} = unknown$ o $s_z.status.alg_{n_i} \in \{candidate, active, victim\}$. Todos esos estados son exigidos para el nodo n_i del nuevo mensaje *AVS* y en consecuencia se verifica la propiedad.
- Si el nodo n_i no coincide con el nodo destino del mensaje *AVS*, n_j, debe comprobarse que el nodo n_i presenta uno de los posibles estados que se dice

en la propiedad para este caso. Al formarse la ruta del mensaje *AVS*, se sabe que el nodo n_i que verifica la propiedad en ese mensaje resulta ser el que la cumple para $s_{z-1}.st\_avsrps_x.path$. Aplicando la hipótesis inductiva, se tiene que $s_{z-1}.status.id_{n_i} = unknown$ o $s_{z-1}.status.alg_{n_i} \in \{candidate, active, victim, dummy\}$. Dado que los efectos de la acción no cambian el valor de *status.alg* y *status.id* de ningún nodo, el nuevo mensaje *AVS* cumplirá este requisito de la propiedad salvo si $s_{z-1}.status.alg_{n_i} = dummy$. En este caso, tras la ejecución de la acción, debe verificarse que $s_z.status.alg_{n_i} = dummy$ y además $s_z.ta_{n_i} = t_i$ y $s_z.blocker_{n_i} = n_{i+1}$. La condición que se impone al nodo n_i de la ruta almacenada en $s_{z-1}.st\_avsrps_x$ cuando su estado es *dummy* es precisamente $s_{z-1}.blocker_{n_i} = n_{i+1}$. Como la variable *blocker* no se ve modificada por los efectos de la acción, sólo será necesario demostrar que en esa situación $s_{z-1}.ta_{n_i} = t_i$ porque la variable *ta* tampoco cambia al ejecutar la acción. Suponiendo que $s_{z-1}.ta_{n_i} \neq t_i$, la propiedad B.2.1 señala que $s_{z-1}.ta_{n_i} < t_i$. Esto implicaría que el nodo n_i, después de incorporarse a la ruta, ejecutó la acción $EndDelArc_{n_i}(n_{i+1})$ o $Abort_{n_i}$ porque son las únicas acciones que incrementan el tiempo de activación del nodo n_i. De acuerdo con la propiedad B.1.6, se descarta que la acción ejecutada sea $Abort_{n_i}$ ya que el nodo n_i, una vez que pasa a estado *aborted*, no puede alcanzar el estado *dummy* en el estado s_{z-1}.

Si se ejecuta $EndDelArc_{n_i}(n_{i+1})$ siendo *dummy* el estado del nodo n_i, el nodo n_i alcanzará el estado *active* con el valor *unknown* en la variable $status.id_{n_i}$ (la propiedad B.2.11 establece que $setPredToInf_{n_i} \neq \emptyset$ porque el antecedente de la propiedad indica que $(n_{i-1}, t_{i-1}) \in s_{z-1}.setPred_{n_i}$). Para que el nodo n_i vuelva a ser *known* precisa la recepción de un mensaje *INF*. Sin embargo, el nodo n_i pasaría a ser *candidate* y no *dummy* como exige el supuesto. Por otra parte, si se ejecuta $EndDelArc_{n_i}(n_{i+1})$ siendo el estado del nodo n_i *candidate*, la propiedad B.2.50 confirma que, mientras $(n_{i-1}, t_{i-1}) \in s_{z-1}.setPred_{n_i}$, el estado final del nodo n_i nunca podrá ser *dummy*. Por consiguiente, se llega a la conclusión de que $s_{z-1}.ta_{n_i} = t_i$ y después de ejecutar la acción analizada la condición temporal sigue cumpliéndose.

Si $x = n_i$ y la propiedad se cumple para el mensaje de $s_{z-1}.st\_avsrsp_x$, siendo $n_i\prime \neq n_j\prime$ y $n_i\prime = n_{1\prime}$, se tiene que $s_{z-1}.blocker_{n_i\prime} = n_2\prime = n_{i\prime+1} \wedge (n_i\prime, t_i\prime) \notin s_{z-1}.setPred_{n_{i\prime+1}}$. En esta situación el nodo x, primero de este mensaje, es el único que pasará a formar parte de la ruta del mensaje *AVS* que verifica la propiedad. Por eso, el mensaje *AVS* que se crea cumple los requisitos que se incluyen para el caso en que $n_i \neq n_j$. Es evidente que el nodo n_i verifica que $s_z.status.alg_{n_i} = candidate$ y éste es un estado de los que indica la propiedad.

- $(sid\prime, t\prime, p\prime, b\prime) \in s.set\_st\_avs_{n\prime} \vee (AVS, sid\prime, t\prime, p\prime, b\prime) \in s.channel(n, n\prime)$

$\Rightarrow$ $first(p\prime) \neq (n_1,\ t_1)$. En caso de que estos mensajes existieran en s_z, se deduce que deberían existir en s_{z-1} porque los efectos de la acción no los generan. Como la estructura de las rutas de esos mensajes no se modifica, de acuerdo con la hipótesis inductiva, se cumple esta condición.

- $sid = s_z.sim\_id_{n_1}$, $s_z.nofirstAVS_{n_1} = false$ y $s_z.st\_avsrsp_{n_1} = NULL$. Al formarse el mensaje *AVS* se elige como campo *sid* el valor que le corresponde a la ruta almacenada en $s_{z-1}.set\_st\_avs_x$ que, por la hipótesis inductiva, se sabe que coincidía con $s_{z-1}.sim\_id_{n_1}$. Dado que además el primer nodo de la ruta del mensaje *AVS* es el primer nodo de la ruta almacenada en $s_{z-1}.set\_st\_avs_x$ es obvio que se verifican todos estos requisitos aplicando la hipótesis inductiva.

Si el nodo x ocupa posiciones posteriores al nodo n_i, la acción no está habilitada porque entonces el nodo x debería pertenecer al camino $ewait(n_i,\ last(p\prime).id,\ p\prime,\ s_{z-1})$ que marca la propiedad. La definición de camino *ewait* indica que los nodos que lo constituyen debe ser *unknown* y la condición de habilitación exige que $s_{z-1}.status.id_x = known$.

- $\pi_z \in \{sndAVSRSP_x\colon x \in \mathcal{N}\}$. El efecto principal de esta acción consiste en la generación de un mensaje *AVSRSP*, a partir del contenido de $s_{z-1}.st\_avsrps_x$ y $s_{z-1}.set\_st\_avs_x$. La hipótesis inductiva permite demostrar que:

  1. $s_z.status.alg_{n_1} = candidate$ ya que el primer elemento de la ruta del nuevo mensaje *AVSRSP* coincide con el primer elemento de la ruta del mensaje contenido en $s_{z-1}.set\_st\_avs_x$.
  2. $p = \gamma_1 \cdot \gamma_2$ con $last(\gamma_1) = (n_i,\ t_i)$. La ruta del nuevo mensaje *AVSRSP* se forma concatenando la ruta del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$ con la de $s_{z-1}.st\_avsrsp_x$ a través del nodo x que tienen en común. La propiedad B.2.21 señala que el par $(x,\ t_x)$ es el último elemento de la ruta del mensaje contenido en $s_{z-1}.set\_st\_avs_x$ y, a su vez, la propiedad B.2.19 indica que el primer elemento de la ruta de $s_{z-1}.st\_avsrps_x$ es $(x,\ s_{z-1}.ta_x)$. Por otra parte, según la propiedad B.2.16, asegura que el penúltimo elemento de la ruta correspondiente a $s_{z-1}.set\_st\_avs_x$ está contenido en $s_{z-1}.setPred_x$. Dado que ésta es la condición del antecedente de la propiedad para el nodo n_i, se concluye que el nodo x coincide con n_i. La hipótesis inductiva permite asegurar que la ruta de $s_{z-1}.st\_avsrps_x$ cuenta también con un nodo que verifica la propiedad y su primer nodo se corresponde con el nodo x. Así que se puede concluir que, para la ruta del mensaje *AVSRSP*, se cumplirá este requisito.

3. Los nodos de la ruta del mensaje *AVSRSP* verifican que, $\forall\ l < i$, $s_z.status.id_{n_l}$ = *known*. La variable *status.id* de los nodos de la ruta de $s_{z-1}.set\_st\_avs_x$, a excepción del nodo x, y los de la ruta de $s_{z-1}.st\_avsrps_x$ es *known*. Como el nodo x aparece en la primera posición de la ruta de $s_{z-1}.st\_avsrps_x$ también se considera *known* aunque está excluido en la propiedad para $s_{z-1}.set\_st\_avs_x$.
4. $\forall\ l$ tal que $1 < l < i$: $s_z.status.alg_{n_l} = dummy \vee (s_z.status.alg_{n_l} = candidate \wedge sid > s_z.sim\_id_{n_l} \wedge s_z.inf\_need_{n_l} = true)$. Asumiendo que estas características se cumplen en s_{z-1} para las rutas involucradas por la hipótesis inductiva, sólo hay que comprobar que el nodo x que sirve de nexo en la unión de las rutas y los que componen la ruta usada de $s_{z-1}.set\_st\_avs_x$ cumplen todas las condiciones. El mensaje *AVSRSP* creado adopta como campo *sid* el que contiene el mensaje almacenado en $s_{z-1}.st\_avsrsp_x$. Esto implica que todos los nodos *candidate* que aparecen en la ruta de $s_{z-1}.st\_avsrsp_x$ seguirán cumpliendo la condición impuesta al campo *sid* en el nuevo mensaje *AVSRSP*. La hipótesis inductiva asegura que el campo *sid* del mensaje de $s_{z-1}.set\_st\_avs_x$ es mayor que el valor de *sim_id* de todos los nodos *candidate* que se indican. Por otra parte, la acción está habilitada cuando $s_{z-1}.cand\_succ_x$ es superior al campo *sid* del mensaje de $s_{z-1}.set\_st\_avs_x$. Considerando la propiedad B.2.20, esta relación se transforma en que $s_{z-1}.st\_avsrps_x.sid$ es superior al campo *sid* del mensaje de $s_{z-1}.set\_st\_avs_x$. En consecuencia, el campo *sid* del mensaje *AVSRSP* es mayor que el valor de *sim_id* de todos los nodos *candidate* que aparecen en la parte de su ruta que procede del mensaje almacenado en $s_{z-1}.set\_st\_avs_x$. Además, el primer elemento de la ruta de $s_{z-1}.st\_avsrsp_x$ es $(x,\ s_{z-1}.ta_x)$ según la propiedad B.2.19 y, de acuerdo con la hipótesis inductiva, $s_{z-1}.st\_avsrsp_x.sid > s_{z-1}.sim\_id_x$ por ser el nodo x el primer nodo de esa ruta. Al ejecutar la acción no se modifica sim_id_x y el mensaje *AVSRSP* adopta como campo *sid* precisamente $s_{z-1}.st\_avsrsp_x.sid$. Esto implica que se cumple que $sid > s_z.sim\_id_x$. Dado que $s_z.inf\_need_x$ pasa a ser *true* y $s_z.status.alg_x$ se mantiene como *candidate*, el nodo x cumple el requisito analizado.
5. $\forall\ l < i$: $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Como el estado de los nodos localizados en las posiciones de las rutas tal que $1 < l < i$ es *dummy* o *candidate*, la propiedad B.2.29 asegura que todos ellos cumplen que $s_z.t\_unk_{n_l} = s_z.ta_{n_l}$. Para aplicar esa propiedad en los nodos *candidate* es necesario que $s_z.st\_alg_l \neq NULL$. Dado que $s_z.inf\_need_{n_l} = true$ para los nodos *candidate*, la propiedad B.2.35 lo confirma. En el caso del nodo ubicado en la primera posición de la ruta del mensaje *AVSRSP*, se recurre a la hipótesis inductiva para concluir que también verifica el requisito temporal. Si el primer nodo de

la ruta almacenada en $s_{z-1}.set\_st\_avs_x$ cumple la condición y los efectos de la acción no introducen cambios ni en la variable *t_unk* ni en *ta* de ningún nodo, el primer nodo de la ruta del mensaje *AVSRSP* también la cumple porque coincide con el de la ruta de $s_{z-1}.set\_st\_avs_x$ considerada.

Al mismo tiempo, el mensaje *AVSRSP* debe cumplir otras características:

- $sid > s_z.sim\_id_{n_1}$. La propiedad B.2.20 establece que, en las condiciones de la acción, $s_{z-1}.st\_avsrsp_x.sid = s_{z-1}.cand\_succ_x$. El campo *sid* del mensaje *AVSRSP* coincide con $s_{z-1}.st\_avsrps_x.sid$. Como la precondición de la acción asegura que $s_{z-1}.cand\_succ_x$ es mayor que el campo *sid* del mensaje de $s_{z-1}.set\_st\_avs_x$ y la hipótesis inductiva confirma que ese valor de *sid* coincide con la identidad simulada del nodo que ocupa la primera posición de su ruta, se tiene que $s_{z-1}.st\_avsrps_x.sid$ es también mayor que la la identidad simulada del primer nodo de esa ruta porque $first(p).id$ es precisamente el primer nodo de la ruta almacenada en $s_{z-1}.set\_st\_avs_x$. Finalmente, resulta que $sid > s_z.sim\_id_{n_1}$ porque $sim\_id_{n_1}$ no cambia de valor al ejecutar la acción.
- Considerando que $n_i = n_j$, se cumple que $s_z.sim\_id_{n_i} \geq sid$ y $s_z.status.alg_{n_i} \in \{candidate, active, victim\}$ o $s_z.status.id_{n_i} = unknown$. El nodo n_i del mensaje *AVSRSP* es el mismo nodo que el que cumple la propiedad para $s_{z-1}.st\_avsrps_x.path$. Además el campo *sid* del mensaje *AVSRSP* es $s_{z-1}.st\_avsrsp_x.sid$. Como la propiedad se cumple en s_{z-1} para la ruta almacenada en $s_{z-1}.st\_avsrps_x$ y los efectos de la acción no cambian el valor de *status.alg*, de *status.id* y de *sim_id* para ese nodo, la propiedad se verifica en s_z.
- Cuando $n_i \neq n_j$ y a la vez $n_i \neq n_1$, el estado del nodo n_i del mensaje *AVSRSP* puede adoptar diferentes valores y se tiene que cumplir en s_z variadas condiciones dependiendo de su estado. En todas las situaciones se concluye aplicando la hipótesis inductiva. Los efectos de la acción no generan ni eliminan mensajes *INF* y tampoco se modifican las características de los mensajes *INF* que podrían existir en s_{z-1}. Además, los requisitos que se cumplían en s_{z-1} para el nodo n_i de la ruta almacenada en $s_{z-1}.st\_avsrsp_x$ se siguen cumpliendo en s_z para el mensaje *AVSRSP* ya que, por la formación de su ruta, ése será el nodo n_i del mensaje. Las condiciones relacionadas con el campo *sid* también se siguen cumpliendo porque el mensaje *AVSRSP* adopta como valor de este campo precisamente $s_{z-1}.st\_avsrsp_x.sid$.
- Si $n_i \neq n_j$ y n_i coincide con el nodo x, siendo éste además el primer nodo de la ruta del mensaje *AVSRSP*, la formación del mensaje *AVSRSP*

no es posible. La razón de ello es que para poder formarse el mensaje, la primera parte de la ruta que procede de la ruta del mensaje almacenado en $s_{z-1}.set\_st\_avs_{n_i}$ tendría que constar de un único elemento y el tratamiento de los mensajes *AVS* impide almacenar mensajes con solamente un elemento.

En el caso en que $n_i \neq n_j$ y el nodo x coincida con el nodo n_i que fija la propiedad para el mensaje almacenado en $s_{z-1}.st\_avsrsp_{n_i}$, se deben analizar los efectos de la acción ya que $s_z.status.alg_{n_i} = candidate$ y $s_z.inf\_need_{n_i} = true$. Asumiendo que el mensaje almacenado en $s_{z-1}.st\_avsrsp_{n_i}$ cumple la propiedad por hipótesis inductiva, se sabe que en s_{z-1} existía un mensaje *INF* directo o indirecto verificando $m_{INF}.sid > s_{z-1}.st\_avsrsp_{n_i}.sid$. Se puede deducir fácilmente que sólo es posible que $p - first(p) \neq s_z.st\_alg_{n_1}$ porque la otra opción requiere que el nodo n_i aparezca en el interior de la ruta de $s_{z-1}.st\_alg_{n_1}$ y su estado sería *dummy* (propiedad B.2.55). Para que la acción esté habilitada $s_{z-1}.status.alg_{n_i} = candidate$ y eso supone que el nodo n_i debería ejecutar una serie de acciones, entre ellas, $EndDelArc_{n_i}(n_{i+1})$. Como esta acción modifica el tiempo de activación del nodo n_i, es evidente que en la rutas en las que apareciera este nodo posteriormente y, antes de ejecutar la acción analizada, tendría un tiempo diferente al que estaba registrado en $s_{z-1}.st\_alg_{n_1}$.

Cuando $p - first(p) \neq s_{z-1}.st\_alg_{n_1}$ y existe un mensaje *INF* en s_{z-1} verificando la propiedad, $m_{INF}.sid > s_{z-1}.st\_avsrsp_{n_i}.sid$. Dado que el nuevo mensaje *AVSRSP* adopta como campo *sid* el valor $s_{z-1}.st\_avsrsp_{n_i}.sid$ y el mensaje *INF* no se modifica, la propiedad se sigue verificando. Por otra parte, se confirma que $sid > s_z.sim\_id_{n_i}$ porque $sid = s_{z-1}.st\_avsrsp_{n_i}.sid$. La propiedad B.2.25 establece que, si $t_{n_i} = s_{z-1}.ta_{n_i}$, $s_{z-1}.st\_avsrsp_{n_i}.sid > s_{z-1}.sim\_id_{first(s_{z-1}.st\_avsrsp_{n_i}.path).id}$. Esto significa que $s_{z-1}.st\_avsrsp_{n_i}.sid > s_{z-1}.sim\_id_{n_i}$. Como los efectos de la acción no modifican estas variables, finalmente, se tiene que, $sid > s_z.sim\_id_{n_i}$.

Cuando el nodo x ocupa posiciones posteriores al nodo n_i, la acción no está habilitada porque entonces el nodo x debería pertenecer al camino $ewait(n_i, last(p').id, p', s_{z-1})$ que marca la propiedad. La definición de camino *ewait* indica que los nodos que lo constituyen deben ser *unknown* y la condición de habilitación exige que $s_{z-1}.status.id_x = known$.

Finalmente, nótese que, tras la ejecución de la acción, el cambio a *true* del último campo del mensaje que permanece almacenado en $s_z.set\_st\_avs_x$ no afecta al cumplimiento de la propiedad.

■

Esta propiedad garantiza que, cuando un nodo se convierte en *victim*, existe un ciclo en el sistema. El estado del resto de nodos del ciclo será *dummy* o *candidate* y, obviamente, conocen su identidad simulada en el momento de la detección.

Propiedad B.2.59. $\exists\ i \in \mathcal{N}$ tal que $s.status.alg_i = victim \Rightarrow \exists\ p \in P$ tal que $wait(n,\ n,\ p,\ s) \wedge \forall\ l \in nodes(p)$ se verifica que $s.status.id_l = known \wedge \forall\ l \in nodes(p) \setminus \{i\}$: $s.status.alg_l = dummy \vee s.status.alg_l = candidate$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_{n_i} = active$ por lo que la propiedad es cierta. Las acciones que afectan a las variables de la propiedad son:

- $\pi_z \in \{StartAddArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. La condición de habilitación de la acción indica que $s_{z-1}.state_x = active$ o, por la propiedad B.3.3, $s_{z-1}.status.alg_x = active$. Así que, ninguno de los nodos señalados por la propiedad puede ejecutar la acción.

- $\pi_z \in \{EndAddArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge\ t \in \mathbb{N}\}$. Para que pueda ejecutarse la acción es necesario que $(y,\ t,\ sent) \in s_{z-1}.set\_waiters_x$. La propiedad B.1.5 permite afirmar que entonces $(y,\ t,\ received) \notin s_{z-1}.set\_waiters_x$ o, lo que es lo mismo, $(y,\ t) \notin s_{z-1}.setPred_x$. Eso implica que el nodo y no está contenido en el ciclo que describe el camino *wait* de la propiedad en s_{z-1}. Si un nodo cualquiera de los que pertenece al ciclo que señala el camino *wait*, ejecuta esta acción, no se modifican ni su correspondiente *status.alg* ni *status.id*. La ejecución de la acción no elimina esperas ni cambia los valores de las variables que tienen que ver con el estado. En consecuencia, la hipótesis inductiva concluye.

- $\pi_z \in \{StartDelArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge\ t \in \mathbb{N}\}$. Suponiendo que $s_{z-1}.status.alg_i = victim$, la acción está habilitada cuando $s_{z-1}.status.alg_y = aborted$ e $(y,\ t) \in s_{z-1}.setPred_x$. En esa situación, se deduce que el nodo y no puede formar parte del ciclo que indica la propiedad. La hipótesis inductiva establece que el estado del resto de los nodos del ciclo sólo puede ser *dummy* o *candidate*. Eso implica que, al ejecutarse la acción, el ciclo no varía y la propiedad se cumple por la hipótesis inductiva.

- $\pi_z = \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Si se supone que $s_{z-1}.status.alg_i = victim$ y el nodo x forma parte del camino p que indica la propiedad entonces, $(x,\ t) \in s_{z-1}.setPred_n$. De acuerdo con la propiedad B.3.2, eso equivale a que $(x,\ t,\ received) \in s_{z-1}.set\_waiters_n$ y, por la propiedad B.1.5, $s_{z-1}.blocker_x = n$. Por otra parte, la condición de habiltación exige que $(x,\ t,\ released) \in s_{z-1}.set\_waiters_y$ y, aplicando la propiedad B.1.5, resulta que $s_{z-1}.blocker_x = y$. Como el nodo x

sólo puede esperar por un único nodo (modelo único recurso) $n = y$. Sin embargo, esto es imposible porque la propiedad B.1.5 indica que si (x, t, *released*) $\in s_{z-1}.set\_waiters_y$ entonces (x, t, *received*) $\in s_{z-1}.set\_waiters_y$.

Suponiendo que la acción queda habilitada porque $s_{z-1}.status.alg_y = aborted$, se hace evidente que en ese caso no existe el camino p que señala la propiedad. De acuerdo con la propiedad B.2.10, para ese estado del nodo y se tiene que $s_{z-1}.setPred_y = \emptyset$. Dado que $s_{z-1}.blocker_x = y$, se concluye que (x, t, *received*) $\notin s_{z-1}.setPred_y$ y, por tanto, los nodos implicados no forman parte de un camino que verifique *wait*.

- $\pi_z \in \{Abort_x: x \in \mathcal{N}\}$. La acción está habilitada cuando el nodo x coincide con el nodo n_i porque $s_{z-1}.status.alg_{n_i} = victim$. Al ejecutarse la acción, $s_z.status.alg_{n_i} = aborted$ y la propiedad pasa a ser cierta con el antecedente falso. Por otra parte, la acción no puede estar habilitada para ninguno de los otros nodos que conforman el ciclo ya que su estado es *dummy* o *candidate*.

- $\pi_z \in \{initiate_x: x \in \mathcal{N}\}$. La condición de habilitación exige que $s_{z-1}.status.alg_x = blocked$. Esto implica que la acción no puede estar habilitada para ninguno de los nodos del ciclo.

- $\pi_z \in \{rcvALG_x(y, m): \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{ALG}\}$. Entre los posibles efectos de esta acción se encuentra el que hace que $s_z.status.alg_x = victim$. En la ruta del mensaje ALG recibido en esta situación existe un camino, $m.path$, que verifica $wait(x, x, m.path, s_z)$. La propiedad B.2.55 asegura que todos los nodos ese camino, a excepción del nodo x, son *dummy*. Por tanto, la propiedad es cierta en s_z.

  Considerando el caso en el que la acción es ejecutada por un nodo del camino *wait* distinto a i, la propiedad B.2.55 asegura la existencia en s_{z-1} de un nuevo ciclo que empieza y acaba en el nodo x que pasa a *victim*. Por otro lado, la propiedad B.2.3 permite confirmar que el nodo i también está incluido en ese nuevo ciclo y, en consecuencia, los ciclos son coincidentes. Aplicando otra vez la propiedad B.2.55, se llega a una contrradicción. El estado de los nodos contenidos en un ciclo que empieza y acaba en el nodo x debe ser *dummy* y el estado del nodo i, según la propiedad era *victim* en s_{z-1}. Por tanto, queda demostrado que la acción sólo se ejecuta cuando $x = i$.

- $\pi_z \in \{rcvINF_x(y, m): \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Para que la acción esté habilitada es necesario que $s_{z-1}.status.id_x = unknown$. En consecuencia, ninguno de los nodos contenidos en el camino *wait* señalado en la propiedad puede ejecutar la acción porque son *known*, incluido el nodo i que, siendo *victim*, la propiedad B.2.15 indica que es también *known*.

- $\pi_z \in \{rcvAVS_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Al ejecutarse la acción podría suceder que $s_z.status.alg_x = victim$. En esa situación, se cumple que el primer nodo de la ruta del mensaje *AVS* y el último coinciden con el nodo x. Según la propiedad B.2.58, la ruta del mensaje *AVS*, *m.path*, verifica que $wait(x, x, m.path, s_z)$.

  Si el nodo x es cualquiera de los otros nodos del camino *wait*, el estado de este nodo también podría convertirse en *victim*. Sin embargo, para ello sería preciso que existiera un nuevo ciclo. La propiedad B.2.3 permite confirmar que el nodo n_i estaría incluido en ese nuevo ciclo y, en consecuencia, los ciclos serían coincidentes. De acuerdo con la propiedad B.2.58, el estado de los nodos contenidos en la ruta del mensaje *AVS* puede ser *dummy* o *candidate*. Eso contradice, el supuesto de partida que establece que $s_{z-1}.status.alg_{n_i} = victim$. Así que, la propiedad queda demostrada.

■

Un interbloqueo es una situación estable hasta que es resuelto con el aborto de un nodo que pertenece a dicho interbloqueo.

Propiedad B.2.60. Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S, $\alpha \in execs(S)$. $\forall\ i \in \mathcal{N}$ se verifica que $s_{z-1}.status.alg_i = victim \Rightarrow s_z.status.alg_i = victim \vee s_z.status.alg_i = aborted$.

Demostración: Las únicas acciones del algoritmo que podrían estar habilitadas y modifican la variable *status.alg* son:

- $\pi_z \in \{EndDelArc_i(j)\colon j\}\ \imath\ \mathcal{N}\}$. Si $s_{z-1}.status.alg_i = victim$, la propiedad B.2.59 asegura que existe un ciclo definido como $wait(i, i, p, s_{z-1})$. La acción está habilitada cuando $s_{z-1}.blocker_i = j$ y, por tanto, el nodo j también debe pertenecer al ciclo. Además, por formar parte del ciclo ambos nodos se tiene que cumplir que $(i, t) \in s_{z-1}.setPred_j$ o $(i, t, received) \in s_{z-1}.set\_waiters_j$ (propiedad B.3.2). Sin embargo, esto impide que la acción esté habilitada porque la propiedad B.1.5 indica que entonces $(i, t, released) \notin s_{z-1}.set\_waiters_j$. Por otra parte, la acción también puede quedar habilitada si $s_{z-1}.state_j = aborted$. En este caso, la definición de ciclo supone que todos los nodos que lo constituyen son *blocked*, haciendo así imposible la ejecución de la acción.

- $\pi_z = Abort_i$. Al ejecutar la acción, $s_z.status.alg_i = aborted$.

- $\pi_z \in \{rcvALG_i(j, m)\colon j \in \mathcal{N} \wedge m \in M_{ALG}\}$ y $\pi_z \in \{rcvAVS_i(j, m)\colon j\} \in \mathcal{N} \wedge m \in M_{AVS}\}$. Los efectos de la acción pueden hacer que $s_z.status.alg_i = victim$ (consecuente cierto). En otros casos, la hipótesis inductiva concluye.

■

Considerando la ejecuciones equitativas del sistema, se puede demostrar que si un nodo i detecta un interbloqueo, $s_z.status.alg_i = victim$, entonces ese nodo finalmente aborta. Esta propiedad asegura que, una vez que se detecta un interbloqueo, éste acabará resolviéndose.

Propiedad B.2.61. Sea $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots$ una ejecución de S, $\alpha \in execs(S)$. $\forall\ i \in \mathcal{N}$ se verifica que $\exists\ z_0$ tal que $s_{z_0}.status.alg_i = victim \Rightarrow \exists\ z_n > z_0$ tal que $s_{z_n}.status.alg_i = aborted$.

Demostración: Teniendo en cuenta la propiedad B.2.60 que indica que, una vez alcanzado el estado *victim*, la acción $Abort_i$ permanece habilitada hasta que sea ejecutada. La propiedad de equidad débil sobre la partición de S concluye que, finalmente $s_{z_n}.status.alg_i = aborted$. ■

Tal y como indica la siguiente propiedad, la acción que permite que una ruta se transforme en un ciclo es la acción $EndAddArc_x((y,\ t)$. Los efectos de esta acción completan la espera que hace que se forme una cadena de esperas cíclicas.

Propiedad B.2.62. Sea $p \in P^+$ que verifica $\mathcal{C}(p,\ s_z) \wedge p \notin cycles(s_{z-1}) \Rightarrow \exists\ x,\ y \in nodes(p)$ tal que $\pi_z = \{EndAddArc_x(y,\ t)\colon t \in \mathbb{N}\}$

Demostración: Las variables que caracterizan a un ciclo según las definiciones del capítulo 3 3.2.1, 3.2.4 y 3.2.6 son: *state*, *blocker* y *set_waiters*. De todas las acciones del algoritmo, sólo es necesario analizar los efectos de las que modifican alguna de esas variables. Las acciones $initiate_x$, $rcvALG_x(y,\ m)$, $rcvAVS_x(y,\ m)$ y $rcvINF_x(y,\ m)$ cambian el valor de la variable *status.alg*, pero en todo esos casos el estado resultante cumple que $s_z.state_x = blocked$ tal y como exige la definición de ciclo. Por otra parte, los efectos de las acciones $firstAVS_x$, $rcvAVSRSP_x(y,\ m)$, $dltINF_x(y,\ m)$, $sndAVS_x$ y $sndAVSRSP_x$ no influyen en las variables que definen un ciclo. En resumen, la ejecución de las acciones mencionadas no permitirán completar un ciclo, a no ser que el ciclo ya estuviera formado antes de su ejecución.

Considerando las acciones que tienen que ver con la creación y borrado de esperas, se observa que ninguna de ellas, a excepción de $EndAddArc_x(y,\ t)$, permite cerrar un ciclo de esperas. A continuación, se describen los efectos de cada una de estas acciones.

- $\pi_z \in Abort_x$. Al ejecutar esta acción $s_z.status.alg_x = aborted$ o, por la propiedad B.3.5, $s_z.state_x = aborted$. En s_z, por definición, la variable *state* de todos los nodos que componen el ciclo debe ser *blocked*. Esto implica que que el nodo x no podría formar parte del ciclo y la ejecución de la acción no afectaría al ciclo.

- $\pi_z \in StartAddArc_x(y)$. En este caso, los efectos provocan que $s_z.blocker_x = y$ y $(x, t, sent) \in s_z.set\_waiters_y$. De acuerdo con la propiedad B.1.5, eso supone que $(x, t, received) \notin s_z.set\_waiters_y$. Por consiguiente, aunque el nodo x pase a estar bloqueado por el nodo y y tenga el estado adecuado, el nodo x no puede formar parte del ciclo porque es necesario que cumpla que $(x, t, received) \in s_z.set\_waiters_y$.

- $\pi_z \in StartDelArc_x(y, t)$. Esta acción elimina el par (y, t) de $setPred_x$, es decir, $(y, t, received) \notin s_z.setPred_x$. Es evidente, por tanto, que el nodo y no puede pertenecer al ciclo.

- $\pi_z \in EndDelArc_x(y)$. La ejecución de la acción conlleva que que $s_z.blocker_x = NULL$. Este efecto impide que el nodo x esté incluido en el ciclo considerado.

- $\pi_z \in EndAddArc_x(y, t)$. Esta acción es la única que puede completar un ciclo tras su ejecución. Sus efectos hacen que la espera ya iniciada entre el nodo x y el nodo y termine de formarse, esto es, $(y, t, received) \in s_z.set\_waiters_x$, $s_{z-1}.blocker_y = s_z.blocker_y = x$ y $s_z.state_y = blocked$. Además, las variables asociadas al nodo x no se ven modificadas y se supone que en s_{z-1} cumplían las condiciones para formar parte del ciclo.

■

En la propiedad que sigue, se asegura que de todos los nodos que pertenecen a un camino *ewait* sólo el último tiene habilitada la receción de un mensaje *INF*, que le permita reconstituir su identidad simulada o propagarla.

Propiedad B.2.63. Sea $\{i, j\} \subseteq \mathcal{N}$, sea $p \in P^+$ tal que $ewait(i, j, p, s)$. $\forall\ k \in \mathcal{N} \setminus \{j\}$: $(j, s.t\_unk_j) \notin s.setPredToInf_k \Rightarrow \exists\ k\prime \in \mathcal{N}$: $(INF, sid, s.t\_unk_j) \in s.channel(k\prime, j)$

Demostración: El camino p puede estar contenido en cualquier ruta almacenada en *st_alg* o en *st_avsrsp* además de en la ruta que viaja en un mensaje *ALG* o *AVSRSP*.

Tanto para la ruta almacenada en *st_alg* como para la ruta incluida en un mensaje *ALG*, la propiedad B.2.54 asegura la existencia de un mensaje *INF* siempre que haya un camino que verifica *ewait*. Dicho mensaje *INF* estará dirigido precisamente al último nodo registrado en el camino p. A continuación, se citan los consecuentes de la propiedad B.2.54 que hay que considerar: consecuente *C1-5* asociado al antecedente *A1*, consecuente *C2-4a* asociado al antecedente *A2*, consecuente *C2-5* asociado al antecedente *A2* y consecuente *C3-1* asociado al antecedente *A3*.

En los consecuentes *C1-5*, *C2-5* y *C3-1*, se asume que existe una ruta $p\prime$ que cumple $ewait(i\prime, last(p\prime).id, p\prime, s_z)$ y también un mensaje $(INF, sid\prime, s_z.t\_unk_{last(p\prime).id})$

$\in s_z.channel(n, last(p\prime).id)$. Dado que $\forall\ k$, $(j, s_z.t\_unk_j) \notin s_z.setPredToInf_k$, el nodo j será el extremo final del camino del *ewait*, o sea, $last(p\prime.id) = j$. Por otra parte, $i\prime = i$, $n = k\prime$ y $s_z.status.id_{i\prime} = unknown$. En el caso del consecuente *C2-4a*, la ruta $p\prime$ cumple la definición de $ewait(j\prime, last(p\prime).id, p\prime, s_z)$. Así que, todo se verifica de igual manera siendo $last(p\prime.id) = j$ y $j\prime = i$.

De acuerdo con la definición de *ewait*, sólo falta estudiar la existencia de un mensaje *INF* viajando hacia el nodo j cuando la ruta p está contenida en $s_z.st\_avsrsp_{n_1}$ o en un mensaje *AVSRSP* que procede del nodo n_k y se dirige al nodo n_1. Las situaciones que se deben analizar, según la propiedad B.2.58, son aquellas en las que los nodos de la ruta p cumplen que:

- $n_i \neq n_j$ y $n_i = n_1$
- $n_i \neq n_j$ y $n_i \neq n_1$, siendo $s_z.status\_id_{n_i} = unknown$ y $s_z.inf\_need_{n_i} = true$ o $s_z.status.alg_{n_i} = candidate$, $s_z.inf\_need_{n_i} = true$ y $s_z.blocker_{n_i} = n_{i+1}$. Cumpliendo en ambos casos que $(sid, p - first(p)) = s_z.st\_alg_{n_1}$.
- $n_i \neq n_j$ y $n_i \neq n_1$, siendo $s_z.status\_id_{n_i} = unknown$ y $s_z.inf\_need_{n_i} = true$ o $s_z.status.alg_{n_i} = candidate$, $s_z.inf\_need_{n_i} = true$ y $s_z.blocker_{n_i} = n_{i+1}$. Siempre que en ambos casos se cumpla que $(sid, p - first(p)) \neq s_z.st\_alg_{n_1}$.
- $n_i \neq n_j$ y $n_i \neq n_1$, siendo $s_z.status.alg_{n_i} = dummy$ y $s_z.blocker_{n_i} = n_{i+1}$ o $s_z.status.id_{n_i} = unknown$ y $s_z.inf\_need_{n_i} = false$). En ambos casos se debe cumplir que$(sid, p - first(p)) = s_z.st_alg_{n_1}$.
- $n_i \neq n_j$ y $n_i \neq n_1$, siendo $s_z.status.alg_{n_i} = dummy$ y $s_z.blocker_{n_i} = n_{i+1}$ o $s_z.status.id_{n_i} = unknown$ y $s_z.inf\_need_{n_i} = false$). Cumpliéndose en ambos casos que $(sid, p - first(p)) \neq s_z.st\_alg_{n_1}$.

En todos esos casos, $n_i = i$, $last(p\prime) = j$ y $n = k\prime$. Como el antecedente de la propiedad establece que $\forall\ k$, $(j, s_z.t\_unknown_j) \notin s_z.setPredToInf_k$ es evidente que el nodo j ocupa la última posición en el camino del *ewait* y la indicada es la única relación entre los nodos mencionados. ■

La siguiente propiedad permite afirmar que los nodos de un camino *ewait*, a excepción del último, no pueden recibir mensajes *INF*. Si existe un mensaje *INF* hacia ellos, la acción $dltINF_x(y, m)$ estará habilitada y llegará a ejecutarse.

Propiedad B.2.64. Sea $\{n_i, n_j\} \subseteq \mathcal{N} \wedge p \in P^+$ tales que $ewait(n_1, n_l, p, s) \wedge p = (n_1, t_1)(n_2, t_2) \ldots (n_l, t_l)$. $\exists\ k \in \mathcal{N}$, $\exists\ m \in M_{INF}$ tal que $\exists\ i < l$: $m \in s.channel(k, n_i) \Rightarrow m.ta \neq s.t\_unk_{n_i}$

Demostración: La propiedad B.2.7 establece que si $(i,\ t) \in s_z.setPredToInf_j$ entonces $\forall\ k \neq j\ \nexists\ m \in M_{INF}$: $m \in s.channel(k,\ i)$ y $m.ta = t$. En este punto, sólo queda demostrar que tampoco existe un mensaje *INF* tal que $m \in s_z.channel(j,\ i)$ y $m.ta = t$. Aplicando la propiedad B.2.6, esto resulta evidente. ■

Con esta propiedad se asegura que, dado un camino de esperas rotas en una ruta *ewait*, se pondrán en marcha los mecanismos adecuados para que el primer nodo de ese camino *ewait*, si vuelve a bloquearse, conozca su nueva identidad simulada y pueda participar en la detección de un posible interbloqueo.

Propiedad B.2.65. Sea $\{n_i,\ n_j\} \subseteq \mathcal{N}$ y $p \in P^+$ que verifica $ewait(n_i,\ n_j,\ p,\ s_z)$. $\forall\ z\prime$: $z\prime \geq z$, $s_{z\prime}.state_{n_i} = blocked \Rightarrow \exists\ z\prime\prime$: $z\prime\prime > z \wedge s_{z\prime\prime}.status.id_{n_i} = known$.

Demostración: Si existe un camino *ewait*, $\forall\ k$: $(n_j,\ s_z.t\_unk_{n_j}) \notin s_z.setPredToInf_k$, la propiedad B.2.63 indica que hay un mensaje *INF*, m, dirigiéndose al nodo n_j con el tiempo correcto. Las acciones del algoritmo para el nodo final del camino *ewait* están deshabilitadas porque, o bien $s_z.status.id_{n_j} = unknown$, o bien $m.ta = s_z.t\_unk_{n_j}$ y $s_z.t\_unk_{n_j} \neq -1$. Para comprobar que $\pi_z \in \{dltINF_{n_j}(y,\ m)$: $\{n_j,\ y\} \subseteq \mathcal{N}\}$ no se puede ejecutar es necesario demostrar que $s_z.t\_unk_{n_j} \neq -1$. Esto resulta evidente porque $m.ta = s_z.t\_unk_{n_j}$ y la propiedad B.2.2 asegura que $1 \leq m.ta \leq s_z.ta_{n_j}$.

Por otro parte, $\pi_z \in \{StartAddArc_{n_j}(y)$: $\{n_j,\ y\} \subseteq \mathcal{N}\}$, $\pi_z \in \{EndAddArc_{n_j}(y,\ t)$: $\{n_j,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{StartDelArc_{n_j}(y,\ t)$: $\{n_j,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$ y $\pi_z \in \{EndDelArc_{n_j}(y)$: $\{n_j,\ y\} \subseteq \mathcal{N}\}$ no modifican las características del camino *ewait* ni la ejecución de las acciones deshabilita la acción $\pi_z \in \{rcvINF_{n_j}(y,\ m)$: $\{n_j,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$.

La propiedad B.2.64 señala que el camino *ewait* no se puede reducir debido a que sus nodos intermedios reciban un mensaje *INF* (todos a excepción de n_j).

Además, se debe analizar el comportamiento del nodo inicial del camino *ewait* ante la ejecución de cualquiera de las acciones del algoritmo. Sólo la acción $StartDelArc_x(y,\ t)$ podría aumentarlo haciendo que el nodo y fuera el nuevo extremo inicial del camino *ewait*. Si sucediera esto, el número de nodos de la ruta p que verifica la aserción *ewait* se incrementaría en uno. Según el antecedente de la propiedad $s_z.state_{n_i} = blocked$ o, por la propiedad B.3.4, $s_z.status.alg_{n_i} = \{blocked,\ dummy,\ candidate,\ victim\}$. Como $s_z.status.alg_{n_i} \neq active$, no habrá ninguna posibilidad de incrementar el camino *ewait* por su inicio ya que $s_z.status.alg_{n_i}$ debe ser *active* para que $StartDelArc_x(y,\ t)$ produzca ese efecto.

Por último, el mensaje *INF* que va al último nodo del camino *ewait*, n_j, finalmente llegará (equidad débil). Como ya se ha explicado, ninguna acción del algoritmo puede deshabilitar esta acción porque el nodo n_j verifica su precondición. Los efectos de $rcvINF_{n_j}(y,\ m)$ disminuirán en una unidad el valor del número de nodos asociado al camino *ewait* (inicialmente este valor era $||nodes(p)|| = j - 1$).

En conclusión, el número de nodos del camino p que cumple *ewait* llegará a tener el valor uno cuando en el camino queden tan sólo dos nodos. Esto supone que la recepción de los correspondientes mensajes *INF* habrá ido reduciendo su valor. El mensaje *INF* que llegue al último nodo en esta situación cambiará su estado a *known* y desaparecerá el camino *ewait*. En z'', el nodo n_i o recibirá un mensaje *INF* que lo convierta en *known* o seguirá siendo *known* si ya lo era cuando se originó el camino *ewait*. ■

Esta propiedad garantiza que todos los nodos de un ciclo, como están bloqueados entre sí, finalmente conocerán su identidad simulada.

Propiedad B.2.66. Sea $\mathcal{C}$ un ciclo en el estado s_z, $\mathcal{C}(p, s_z)$, $\forall\ n \in nodes(p) \wedge \forall\ z' > z$ se verifica que $s_{z'}.state_n = blocked \Rightarrow \exists\ z''$: $z'' > z \wedge \forall\ n \in nodes(p)$, $s_{z''}.status.id_n = known$.

Demostración: Si el nodo x pertenece al ciclo $\mathcal{C}$ y $s_z.status.id_x = unknown$ significa que se ejecutó la acción $EndDelArc_x(y)$. Esa acción se pudo ejecutar antes o después de la formación del ciclo.

Si se supone que la ejecución de la acción es posterior a la creación del ciclo, entonces el nodo x tiene que cumplir la precondición de la acción $EndDelArc_x(y)$. Para que esta acción esté habilitada es necesario que $blocker_x = y$ y bien, $(x,\ t,\ released) \in s_z.set\_waiters_y$ o bien, $s_z.state_y = aborted$.

Se sabe que todos los nodos de un ciclo están bloqueados y, además, un nodo sólo puede estar bloqueado por un único nodo. Así que, si $s_z.blocker_x = y$, $y \in nodes(p)$ y $s_z.state_y = blocked$. Por tanto, es obvio que verifica la condición de habilitación que exige que $s_z.state_y = aborted$. La otra posibilidad de habilitación requiere que $(x,\ t,\ released) \in s_z.set\_waiters_y$. La propiedad B.1.4 establece que, en esa situación, $(x,\ t,\ received) \notin s_z.set\_waiters_y$. Según la propiedad B.3.2, eso equivale a $(x,\ t) \notin s_z.setPred_y$. Como el nodo x y el nodo y pertenecen ambos al ciclo $\mathcal{C}(p,\ s_z)$, es evidente que existe una espera $dwait(x,\ y,\ s_z)$ y, por lo tanto, $(x,\ t) \in s_z.setPred_y$. Esto contradice la precondición de la acción analizada. De este razonamiento se deduce que la acción $EndDelArc_x(y)$ sólo puede ejecutarse antes de que tenga lugar la formación del ciclo o interbloqueo considerado.

Se podría pensar que el nodo que envió el mensaje ALG es precisamente $s_z.blocker_x$ en el ciclo. Dado que se ha demostrado que la acción $EndDelArc_x(y)$ se ejecuta antes de que se forme el interbloqueo, el tiempo de activación asociado al nodo x se incrementará como efecto de la acción $EndDelArc_x(y)$ antes de formar parte del ciclo. Esto implica que el tiempo de activación del nodo x en el ciclo no se corresponde con el tiempo con el que está registrado en el camino *ewait*. Por consiguiente, este caso particular también queda demostrado con la misma argumentación que para otros nodos.

Si en el ciclo existe un nodo *unknown*, la propiedad B.2.31 establece que ese nodo tiene almacenada una ruta en st_alg_x y $t\_unknown_x \neq ta_x$. Obviamente, la ruta almacenada se extrajo de un mensaje de tipo ALG que llegó al nodo x. El nodo que envió ese mensaje ALG a x incorporó a x en su correspondiente *setPredToInf*. Si en el momento en el que el nodo x forma parte de un ciclo de nodos interbloqueados, $state_x$ = *blocked*, el nodo x seguirá perteneciendo a *setPredToInf*. La propiedad B.2.54 indica para el caso del antecedente *A-3* que existe un mensaje *INF* directo o indirecto. La recepción del mensaje *INF* directo convierte en *known* al nodo x. Para el mensaje *INF* indirecto, la propiedad B.2.65 concluye que este nodo finalmente también llegará a ser *known*. Así que, finalmente, todos los nodos del ciclo pasan a ser *known* tal y como indica la propiedad. ■

En esta propiedad se describe la forma de detección de un ciclo mediante un mensaje ALG. Si ningún nodo del ciclo es *victim*, habrá uno que sea *candidate* y la existencia de un mensaje ALG, en cuya ruta está registrado el identificador del nodo *candidate*, estará asegurada.

Propiedad B.2.67. Sea $\mathcal{C}$ un ciclo en el estado s_z, $\mathcal{C}(p, s_z)$, $\forall\ n \in nodes(p)$ se tiene que: $s_z.status.alg_n \neq victim \wedge \forall\ z\prime > z, \mathcal{C}(p, s_{z\prime}) \Rightarrow \exists\ z\prime\prime \geq z \wedge \exists\ i \in nodes(p)$: $s_{z\prime\prime}.status.alg_i = candidate \wedge \exists\ \{x, n\} \subseteq nodes(p)$: $\exists\ m \in M_{ALG}$ tal que $m \in s_{z\prime\prime}.channel(n, x) \wedge m.ta = s_{z\prime\prime}.ta_x \wedge (x, t) \in s_{z\prime\prime}.setPred_n \wedge m.path = \gamma_1 \cdot (i, t) \cdot \gamma_2$, siendo $\gamma_1, \gamma_2 \in P^*$

Demostración: Si se supone que, de acuerdo la propiedad B.2.62, el ciclo se cierra al ejecutar la acción $EndAddArc_x(y, t)$, siendo $\{x, y\} \subseteq p$, el estado del nodo x puede ser:

- $s_{z-1}.status.alg_x \in \{dummy, candidate\}$, los efectos de la acción provocan que se envíe un mensaje ALG al nodo y.

  Caso *dummy*: El mensaje ALG que se envía al nodo y tiene como ruta el contenido de $s_{z-1}.st\_alg_x$. El mensaje ALG, a su vez verifica que $m \in s_{z\prime\prime}.channel(x, y)$, $m.ta = s_{z\prime\prime}.ta_y = t$ y $(y, t) \in s_{z\prime\prime}.setPred_x$. Para afirmar que la propiedad se cumple es necesario comprobar que un nodo de los del ciclo es *candidate*. Se sabe que el nodo x no lo es, pero el mensaje ALG almacenado en $s_{z-1}.st\_alg_x$ surgió, a su vez, de la recepción de un mensaje ALG. Ese mensaje ALG lo mandó un nodo que pasó a *candidate* al hacer el envío o lo reenvió un nodo *blocked* que se convirtió en *dummy*. En este último caso, el proceso de reenvío se ha repetido tantas veces como nodos *dummy* se encuentren en el ciclo en s_z. Sabiendo que el ciclo es de tamaño finito, se alcanzará el nodo que ejecutó la acción *initiate* y lo transformó en *candidate*.

- Si, cuando el nodo n mandó el mensaje ALG inicial en s_{z_1} ($z_1 < z$), éste ya formaba parte de las esperas que constituyen el ciclo tal y como aparece en s_z, se sabe que $last(s_z.st\_alg_x.path).id = n$. Así que, siguiendo las esperas del ciclo que están recogidas en la ruta almacenada en st_alg_x se encontrará al nodo n del ciclo que será el nodo *candidate* que haga cumplirse la propiedad. Desde que este nodo, n, *candidate* envió el mensaje no ha cambiado su estado porque el ciclo de esperas no se ha visto modificado o no ha tenido cambios que incluyan las esperas que parten del nodo x y llegan hasta el nodo n. En otras palabras, el nodo n no ha ejecutado la acción $EndDelArc_n(n\prime)$ y, por lo tanto, $s_z.blocker_n = s_{z_1}.blocker_n$. Como cada nodo sigue bloqueado por el mismo nodo que está registrado en el ciclo en s_z y su tiempo tampoco se ha modificado, el mensaje ALG que circule entre dos nodos cualquiera del ciclo será correcto y podrá ser procesado. Además, los distintos mensajes cumplirán que $last(m.path).id = n$ porque se construyen a partir del mensaje ALG recibido, $m\prime$, y $last(m.path).id = last(m\prime.path).id$. De aquí se deduce que, que el mensaje ALG que se propaga como efecto de la acción $EndAddArc_x(y,t)$ tiene la forma $m.path = \gamma_1 \cdot (n,\ t) \cdot \gamma_2$, siendo $\gamma_2 = \epsilon$.
- Sin embargo, cuando el nodo n que envió el mensaje ALG inicial en s_{z_1} no pertenece al ciclo formado en s_z, siendo $z_1 < z$, surge una situación distinta a pesar de que $last(s_z.st\_alg_x.path).id = n$. Las esperas que aparecen reflejadas en $s_z.st\_alg_x$ no se corresponden con las que contiene el ciclo en s_z y, por tanto, el nodo n ejecutó $EndDelArc_n(n\prime)$ en s_{z_2}, donde $z < z_2 < z_1$, para su activación.

  Si, en s_{z_2}, el nodo n es *active* y *known* habrá enviado un mensaje de tipo INF al ejecutar $StarDelArc_n(n\prime\prime,\ t_{n\prime\prime})$ en s_{z_3} ($z < z_3 < z_2$). Desde el nodo del ciclo en el que las esperas ya no se corresponden con las que recorrió el mensaje ALG enviado por el nodo n, esto es, n_i hasta $n\prime\prime$ se habrá formado un camino $ewait(n_i,\ n\prime\prime,\ p,\ s_{z_3})$. Este camino habrá ido reduciendo conforme se haya recibido el mensaje INF correspondiente. Durante el tiempo que el camino *ewait* disminuye, el ciclo $\mathcal{C}$ se puede haber ido formando parcial o totalmente.

  En este punto cabe recordar que la propiedad B.2.66 asegura que en un ciclo finalmente todos los nodos llegan a ser *known*. El estado del nodo n_i fue *active* y *unknown*, convirtiéndose en *blocked* y *unknown* si inicia la espera por el nodo que le bloquea en el ciclo. Al recibir el mensaje INF que esperaba, el nodo n_i habrá pasado directamente a *candidate*.

  En el caso de que el nodo n_i recibiera en primer lugar el mensaje INF (*active* y *known*) y luego se bloqueara por el nodo que le sigue en el ciclo $\mathcal{C}$, los efectos de $StartAddArc_{n_i}(n_{i+1})$ tranformarían también al nodo n_i en

candidate. Como se observa en ambas situaciones, se ha localizado un nodo del ciclo que es *candidate* y se asegura la presencia de un mensaje ALG que cumple los requisitos marcados en el consecuente. En consecuencia, la propiedad quedaría ya demostrada.

El mensaje ALG que apareció como efecto de la acción $rcvALG_x(y,\ m)$ se habrá reenviado por los nodos que se fueron incorporando hasta formar el ciclo en s_z, incluido el nodo n_i. Si el nodo n_i no ha recibido aún el mensaje INF que le pase a *known*, el mensaje ALG seguirá a la espera de ser recibido. Si, por el contrario, el nodo n_i ya es *candidate* y *known* por la recepción del mensaje INF, la acción $rcvALG_n(y\prime,\ m)$ se habrá ejecutado y obviamente la ruta de ese mensaje ALG contiene al nodo n_i, $m.path = \gamma_1 \cdot (n_i,\ s.ta_i) \cdot \gamma_2$, donde $\gamma_1\ ,\ \gamma_2 \neq \epsilon$.

Teniendo en cuenta la posibilidad de que el nodo n en s_{z_2} fuera *active* y *unknown* tras ejecutar $EndDelArc_n(n\prime)$ en s_{z_2} $(z < z_2 < z_1)$, resulta evidente que el camino *ewait* lo contendrá. Así pues, existirá $ewait(n_i,\ n_f,\ p,\ s_{z_2})$, donde $n_i \in nodes(p)$ tal que $\mathcal{C}(p,\ s_z)$. El nodo n_i no puede coincidir con el nodo x porque el nodo x debe mantener su estado a *dummy* desde el momento que recibió el mensaje ALG que tiene almacenado en $s_z.st\_alg_x$ hasta el mismo instante en que se cierra el ciclo $\mathcal{C}$. El proceso en el que desaparece el camino *ewait* y se forma el ciclo $\mathcal{C}$ es similar al descrito anteriormente. Del mismo modo, surge la aparición de un nodo *candidate* que, más concretamente, es el nodo n_i. Al igual que en el caso previo, el mensaje ALG verifica la propiedad porque su ruta es de la forma $m.path = \gamma_1 \cdot (n_i,\ s.ta_i) \cdot \gamma_2$, donde $\gamma_1,\ \gamma_2 \neq \epsilon$.

Caso *candidate*: El nodo i de la propiedad coincide con x y el mensaje ALG verifica que $m \in s_{z\prime\prime}.channel(x,\ y)$, $m.ta = s_{z\prime\prime}.ta_y = t$, $(y,\ t) \in s_{z\prime\prime}.setPred_x$ y $m.path = (x,\ s_{z\prime\prime}.ta_x)$. Por tanto, la propiedad, se cumple ya que $\gamma_1 = \gamma_2 = \epsilon$.

- $s_{z-1}.status.alg_x = victim$. Según el antecedente de la propiedad esta opción queda excluida.

- $s_{z-1}.status.alg_x \in \{active,\ aborted\}$. Si se forma un ciclo, todos su nodos serán *blocked* por definición. El nodo x no puede ejecutar la acción $EndAddArc_x(y,\ t)$ en este estado porque los efectos de la acción no cambian la variable $status.alg_x$.

- $s_{z-1}.status.alg_x = blocked$ y $s_{z-1}.status.id_x = known$. La acción $EndAddArc_x(y,\ t)$ cierra el ciclo. Como el estado se mantiene, el nodo x tiene habilitada, bien la acción $initiate_x$, o bien la acción $rcvALG_x(y,\ m)$. Estas acciones no pueden ser deshabilitadas por otras acciones del algoritmo distintas de ellas. Por una parte, $StartAddArc_x(y)$ requiere que $status.alg_x = active \neq blocked$, $Abort_x$

sólo se ejecuta si $status.alg_x = victim \neq blocked$ y $rcvINF_x(y, m)$ se ejecuta cuando $status.id_x = unknown \neq known$. Las acciones $firstAVS_x$, $sndAVS_x$, $sndAVSRSP_x$ están habilitadas si $status.alg_x = candidate \neq blocked$. Si se ejecuta $EndDelArc_x(y)$ significa que previamente se ha roto el ciclo de esperas, esto es $(y, t, received) \notin setPred_x$, violando el antecedente. Con la acción $rcvAVS_x(y, m)$, el nodo x puede llegar a ser *victim* siendo inicialmente $candidate \neq blocked$ o por el contrario no cambia el estado del nodo x.

Para el resto de acciones, como no se modifica la variable $status.alg_x$, se tiene que es posible todavía ejecutar $initiate_x$ o $rcvALG_x(y, m)$. La partición y la equidad débil aseguran que cualquiera de las acciones llegará a ejecutarse. Al ejecutarse finalmente las acciones que están habilitadas sucede lo siguiente:

- Los efectos de la acción $initiate_x$ cambian el estado, $status.alg_x = candidate$, y la generación de un mensaje ALG tal que $m \in s_{z\prime\prime}.channel(x, y)$, $m.ta = s_{z\prime\prime}.ta_y = t$, $m.path = (x, s_{z\prime\prime}.ta_x)$ y $(y, t) \in s_{z\prime\prime}.setPred_x$. Se concluye, por tanto, que la propiedad se cumple con $\gamma_1 = \gamma_2 = \epsilon$.
- La acción $rcvALG_x(y, m)$ convierte el nodo x en *dummy*, almacena el mensaje en $s_{z\prime\prime}.st\_alg_x$ y reenvía el mensaje ALG que recibió incorporándose convenientemente a su ruta: $m\prime \in s_{z\prime\prime}.channel(x, n)$, $m\prime.path = (x, s_{z\prime\prime}.t_x).m.path$, $m\prime.ta = s_{z\prime\prime}.ta_n = t$ y $(n, t) \in s_{z\prime\prime}.setPred_x$. La búsqueda de un nodo del ciclo que sea *candidate* supone el mismo razonamiento que ya se apuntó para el caso $s_z.status.alg_x = dummy$ porque éste contiene una ruta en $s_{z\prime\prime}.st\_alg_x$.

- $s_{z-1}.status.alg_x = blocked$ y $s_{z-1}.status.id_x = unknown$. La acción $EndAddArc_x(y, t)$ cierra el ciclo y mantiene el estado del nodo x. La propiedad B.2.66 establece que el nodo x llegará a ser *known*. Eso quiere decir que se puede aplicar lo argumentado para el caso $status.alg_x = blocked$ y $status.id_x = known$.

■

Esta función permite calcular el número de nodos que tovía no se han reconocido como integrantes del ciclo.

Definición B.2.3. Sea $\mathcal{C} \in cycles(s)$ y $p \in \in P^+$. Se define la función *visited_cycle*: $cycles(\mathcal{N}) \times P^+ \times states(S) \rightarrow \mathbb{N}$ tal que $visited\_cycle(\mathcal{C}, p, s) = size\_cycle(\mathcal{C}) - ||visited\_nodes(p, s)||$

Esta propiedad demuestra que la existencia de un único nodo candidato en el ciclo determina que, tarde o temprano, será elegido víctima.

Propiedad B.2.68. Sea $p \in P^+$ un ciclo en el estado s_z, $\mathcal{C}(p, s_z)$. $\forall\ z\prime > z$: $\exists\ i \in nodes(p)$ tal que $\forall\ n \in nodes(p) \setminus \{i\}$, $s_{z\prime}.status.alg_n \neq candidate \Rightarrow \exists\ z\prime\prime > z$: $s_{z\prime\prime}.status.alg_i = victim$.

Demostración: Los nodos de un ciclo, por definición, cumplen que su variable *state* es *blocked*. Si para algún nodo i $s_{z\prime}.status.alg_i = victim$, la propiedad es cierta. Sin embargo, si no existe ningún nodo del ciclo que sea *victim*, la propiedad B.2.67 establece que habrá un nodo que llegue a ser *candidate*. Como el antecedente de la propiedad impone que el estado de todos los nodos, excepto uno, debe ser distinto de *candidate*, se hace evidente que ese nodo *candidate* es el que señala el antecedente de la propiedad y tiene que ser el nodo i que, finalmente, se convertirá en *victim*. Para ello, el nodo *candidate* tiene que recibir un mensaje $(ALG, sid, s_{z\prime\prime}.ta_i, \gamma_1{\cdot}(i, t){\cdot}\gamma_2) \in s_{z\prime\prime}.channel(first(\gamma_1).id, i)$, cumpliéndose que $s_{z\prime\prime}.status.id_i = known$, $s_{z\prime\prime}.status.alg_i = candidate$ y $last(\gamma_1) \in s_z\prime\prime.setPred_i$.

Si existe un ciclo la propiedad B.2.67 establece que habrá un momento en el que haya un mensaje ALG circulando entre cualquier par de nodos del mismo y un nodo i en estado *candidate*. Comprobando que ese mensaje ALG es recibido por el único nodo *candidate* del ciclo y que satisface las características que permite que el nodo i sea *victim*, la propiedad quedará demostrada.

El mensaje que circula por el ciclo, de acuerdo con la propiedad B.2.67, tiene la siguiente estructura: $(ALG, sid, s_{z\prime\prime}.ta_x, \gamma_1{\cdot}(i, t){\cdot}\gamma_2) \in s_{z\prime\prime}.channel(n, x)$ y además $(x, s_{z\prime\prime}.ta_x) \in s_{z\prime\prime}.setPred_n$. Ninguna acción puede deshabilitar la acción $rcvAlG_x(n, m)$. Todos los nodos del ciclo, a excepción del nodo i, son *blocked* o *dummy*. Que un nodo del ciclo sea *dummy* implica que ya ha recibido el mensaje ALG considerado. De acuerdo con la propiedad B.2.29, la variable *t_unk* de un nodo *dummy* coincide con su tiempo de activación. En esta situación, la propiedad B.2.46 garantiza que no existe ningún mensaje ALG con tiempo correcto dirigido al nodo i. Así que los nodos del ciclo que tienen habilitada la acción $rcvALG_x(n, m)$ son *blocked* o *candidate* y su correspondiente *st_alg* está vacío. Esto implica que el mensaje finalmente será retirado por un nodo del ciclo que, o bien se convertirá en *dummy*, o bien alcanzará el nodo i que es *candidate* y que originó el mensaje ALG inicial. En el primer supuesto, cuando el mensaje ALG llega a un nodo *blocked* y lo transforma en *dummy*, la función que registra el número de nodos del ciclo por los que ha pasado el mensaje ALG disminuirá en uno, $visited\_cycle(\mathcal{C}, p, s) = size\_cycle(\mathcal{C}) - visited\_nodes(p)$. Antes de completarse la formación del ciclo, el mensaje ALG ya ha podido recorrer parte de los nodos del mismo. El estado de esos nodos será *dummy* y no cambiará mientras persista el ciclo. Así que, la función $visited\_cycle(p)$ tendrá un valor inferior al que tiene la función en un estado inicial en el que el ciclo está cerrado y todavía no ha llegado el primer mensaje ALG (valor máximo: $visited\_cycle(p) = size\_cycle(\mathcal{C})$).

Por otra parte, ese nodo *dummy* reenvía el mensaje ALG al nodo predecesor, que también forma parte del ciclo, con la ruta modificada, $m\prime$: $(ALG, sid, s_{z\prime\prime}.ta_k, (x,$

$s_{z''}.ta_x)\cdot m.path) \in s_{z''}.channel(x,\ k)$, siendo $(k,\ t_k) \in s_{z''}.setPred_x$. Sabiendo que los nodos x y k pertenecen al ciclo y que las esperas del mismo no han variado, resulta obvio que $(k,\ t_k) \in s_{z''}.setPred_x$ por la propia definición de ciclo. Además, la nueva ruta del mensaje ALG contiene el par $(i,\ t)$ porque ésta se construye a partir de $m.path$ que ya lo contenía. En resumen, el mensaje ALG reenviado sigue cumpliendo las características necesarias para su correcta propagación. Este proceso se repite hasta que el nodo predecesor del emisor del mensaje ALG sea precisamente el nodo i que es *candidate*.

Como el tamaño del ciclo es finito, en un instante indeterminado la función $visited\_cycle(p)$ tendrá el valor 1 y estará habilitada la acción $rcvALG_i(n,\ m)$. El nodo n será el úlimo nodo del ciclo que haya pasado a *dummy* y el mensaje ALG que éste puso en el canal es exactamente $(ALG,\ sid,\ s_{z''}.ta_i,\ \gamma_1\cdot(i,\ t)\cdot\ \gamma_2) \in s_{z''}.channel(n,\ i)$, siendo $(i,\ t_i) \in s_{z''}.setPred_n$. Comparando con los requisitos que se deben cumplir para que el efecto de la acción $rcvALG_i(n,\ m)$ sea $status.alg_i = victim$, se concluye que la propiedad es correcta porque:

- $s_{z''}.status.alg_i = candidate$ y la propiedad B.2.15 indica que entonces $s_{z''}.status.id_i = known$.
- $last(\gamma_1) \in s_{z''}.setPred_i$. Como $last(\gamma_1).id$ es un nodo del ciclo y la espera entre $last(\gamma_1).id$ y el nodo i no ha cambiado, la definición de ciclo basta para confirmarlo.
- $n = first(\gamma_1).id$. La propiedad B.2.55 asegura que el origen del mensaje ALG que llega al nodo *candidate* coincide con el primer elemento de la ruta del mensaje.

■

En la siguiente propiedad se garantiza que dos nodos candidatos que forman parte de una ruta, que se corresponde con relaciones de espera reales, llegarán a intercambiar información mediante un mensaje de tipo ALG. El mensaje recibido procederá del candidato que ocupe una posición más alejada en la ruta e incluirá la identidad simulada que posea ese nodo candidato.

Propiedad B.2.69. Sea $p \in P^+$ tal que $p = (n_1,\ t_1)(n_2,\ t_2)\ldots(n_m,\ t_m)$. Se verifica que $\exists\ \{i,\ j\} \subseteq \mathbb{N}$ con $i < j \wedge \exists\ z$ tal que $\forall\ z' \geq z$, $wait(n_1,\ n_m,\ p,\ s_{z'}) \wedge s_{z'}.status.alg_{n_i} = s_{z'}.status.alg_{n_j} = candidate \wedge s_{z'}.st\_alg_{n_i} = NULL \wedge \forall\ k$: $i < k < j\ s_{z'}.status.alg_{n_k} \neq \{candidate,\ victim\} \Rightarrow \exists\ z'' \geq z$: $(ALG,\ s_{z''}.ta_{n_i},\ (s_{z''}.sim\_id_{n_j}.id,\ s_{z''}.sim\_id_{n_j}.ta,\ nu),\ \gamma_1\cdot(n_j,\ t_j)\cdot\gamma_2) \in s_{z''}.channel(n_{i+1},\ n_i)$, siendo $nu \in \mathbb{N}^* \wedge \{\gamma_1,\ \gamma_2\} \subseteq P^*$.

Demostración: Como el nodo n_i y el nodo n_j son nodos *candidate*, se sabe que, han enviado un mensaje ALG cuya ruta tenía un único elemento. Las acciones que convierten un nodo en *candidate* son: $StartAddArc_x(y)$, $initiate_x$ y $rcvINF_x(y, m)$. Si n_j es *candidate* debido a la ejecución de las acciones $StartAddArc_{n_j}(y)$ o $rcvINF_{n_j}(y, m)$, el envío de un mensaje ALG está condicionado a tener elementos en $setPred_{n_j}$ que no pertenezcan a $setPredToInf_{n_j}$. Según la existencia del camino que verifica *wait* y del mensaje ALG que ha enviado n_j, se plantean dos posibilidades:

- El camino *wait* ya estaba formado cuando el nodo n_j emitió el mensaje ALG. Esta situación implica que el nodo n_j era *dummy* inicialmente. Para poder

  después se activó y, al ejecutar las acciones indicadas, pasó a ser *candidate* sin enviar ningún nuevo mensaje ALG al nodo predecesor que aparece en el camino *wait* considerado. Si el nodo n_j mandara un mensaje ALG sería a un nodo ajeno al camino *wait* analizado. En el proceso de activación del nodo n_j, cambió su tiempo de activación y la ruta del mensaje ALG que se dirige al candidato n_i del camino p deja de verificar la aserción *wait* a partir precisamente de n_j.

- El camino *wait* no está formado en el momento en el que el nodo n_j reenvía un mensaje ALG. En este caso el nodo n_j también era *dummy*, pero al activarse y pasar a *candidate*, envió un nuevo mensaje ALG formado por un único elemento, su identidad y su tiempo de activación correcto, a un nodo n_{j-1} que se ha bloqueado por él.

En ambos casos, la activación del nodo n_j hizo que éste pasara a ser *unknown*, quedando pendiente la recepción de un mensaje INF para adoptar la identidad simulada del nodo origen que envió el mensaje ALG y que retransmitió a sus predecesores, incluido o no el que aparece en el camino *wait*. Además de modificar la identidad simulada del nodo, la recepción del mensaje INF tiene como efecto el cambio a *candidate-known* del nodo n_j tal y como se requería por el enunciado de la propiedad.

Cuando se recibe el mensaje INF, $sim\_id_{n_j}$ se carga con la identidad del nodo *candidate* que originó el mensaje ALG y retransmitió siendo *dummy*.

La segunda opción planteada es similar a la que se tiene si el camino *wait* ya está constituido y el nodo n_j ejecuta la acción $initiate_{n_j}$. El nodo n_j también es el creador del mensaje ALG que circula por el camino *wait*. Del mismo modo que en ese caso, la ruta del mensaje ALG inicialmente se corresponde con un par que contiene la identidad y el tiempo de activación del nodo n_j. Debido a esta similitud, se englobará el último supuesto de las acciones $StartAddArc_{n_j}(y)$ o $rcvINF_{n_j}(y, m)$ en el análisis de la acción $initiate_{n_j}$.

Considerando la ejecución de la acción $initiate_{n_j}$, se sabe que el mensaje que se genera por efecto de la misma contiene (n_j, t_j, ϵ) como identidad simulada. El mensaje ALG correspondiente al nodo n_j tiene como destino, entre otros, el nodo

que aparece inmediatamente antes en el camino p cumpliendo *wait* y que, por tanto, pertenece a $setPred_{n_j}$. El mensaje ALG formado responde al siguiente formato: (ALG, t_k, $s_{z\prime}.sim\_id_{n_j}$,(n_j, $s_{z\prime}.ta_{n_j}$)) $\in$ $s_{z\prime}.channel(n_j, k)$, siendo ($k$, t_k) $\in$ $s_{z\prime}.setPred_{n_j}$. Como el nodo n_k forma parte también del camino p que cumple $wait(n_1, n_m, p, s_{z\prime})$, se asume que $t_k = s_{z\prime}.t\_activ_{n_k} = s_{z\prime}.ta_{n_k}$.

Como el nodo $s_{z\prime}.state_{n_k} = blocked$, los posibles estados que puede presentar, teniendo en cuenta el antecedente de la propiedad, son *blocked* y *dummy*. Se puede afirmar que todo nodo *dummy* del camino p que cumple la definición *wait* ya ha recibido un mensaje ALG. De acuerdo con la propiedad B.2.29, la variable *t_unk* de un nodo *dummy* coincide con su tiempo de activación. En esta situación, la propiedad B.2.46 garantiza que no existe ningún mensaje ALG con tiempo correcto dirigido al nodo n_k. Así que los nodos del camino *wait* que tienen habilitada la acción $rcvALG_x(n, m)$ son *blocked* o *candidate* con su *st_alg* vacío.

Así que, todos los nodos existentes entre los nodos *candidate*, n_i y n_j, dejarán pasar el mensaje ALG inicial convirtiéndose en nodos *dummy*. Esto es consecuencia de que todos los nodos n_k, a los que hace referencia el antecedente de la propiedad son *blocked*, tienen habilitada la acción $rcvALG_{n_k}(n_{k\prime}, m)$. Por la propiedad de equidad débil sobre la partición, la acción finalmente será ejecutada. Al mismo tiempo, ninguna otra acción del algoritmo puede deshabilitar la acción $rcvALG_x(y, m)$ para los nodos n_k, mientras persista el camino *wait*:

- Por una parte, $StartAddArc_x(y)$ requiere que $status.alg_x = active \neq blocked$, $Abort_x$ sólo se ejecuta si $status.alg_x = victim \neq blocked$ y $rcvINF_x(y, m)$ se ejecutará cuando $status.id_x = unknown \neq known$. Las acciones $firstAVS_x$, $sndAVS_x(y)$, $sndAVSRSP_x(y)$ están habilitadas si $status.alg_x = candidate \neq blocked$.

- Si se ejecuta $EndDelArc_x(y)$ significa que previamente se ha roto el camino *wait* de esperas, esto es (y, t, *received*) $\notin$ $set\_waiters_x$.

- Con la acción $rcvAVS_x(y, m)$, el nodo x puede llegar a ser *victim* siendo inicialmente *candidate* $\neq$ *blocked* o por el contrario no cambia el estado del nodo x.

- Para el resto de acciones, como no se modifica la variable $status.alg_x$, es posible todavía ejecutar $rcvALG_x(y, m)$.

- Al ejecutar $initiate_x$, el nodo x pasa a ser *candidate* y el requisito marcado para los nodo n_k deja de verificarse.

El efecto de esta acción para un nodo n_k supone el reenvío de un mensaje ALG en el que sólo se modifica la ruta y el tiempo del mensaje (tiempo del nodo destino del mensaje). La ruta va creciendo porque los nodos *blocked* se anotan en la primera

posición de la misma, manteniéndose como $last(m.path) = (n_j, t_j)$. El campo $m.sid$ se copia del campo que lleva consigo el mensaje ALG recibido. Repitiendo sucesivamente el reenvío de mensajes ALG entre los nodos *blocked* existentes, llega un momento en el que el destino del mensaje es el nodo n_i que es *candidate* y tiene su correspondiente $st\_alg_{n_i} = NULL$. Este nodo también tiene habilitada la acción $rcvALG_{n_i}(n_k, m)$ y por la propiedad de equidad débil sobre la partición, finalmente se ejecutará.

Para un mensaje ALG generado por la acción $initiate_{n_j}$ que discurre entre los nodos *candidate*, n_i y n_j, la identidad simulada del nodo n_j que aparece en el mensaje ALG no se modifica. Para que cambiara su valor, el nodo n_j debería activarse. Como se supone que el nodo n_j pertenece al camino *wait* en $s_{z''}$, el estado del mismo se mantiene a *candidate* y no se activa.

Si el mensaje ALG se crea antes de que el nodo n_j se convierta en *candidate*, las identidades simuladas no se controlan tan fácilmente y pueden variar de acuerdo a la evolución del estado del nodo n_j. Las acciones que modifican la identidad simulada de un nodo son $rcvINF_x(y, m)$, $StartDelArc_x(y, t)$ y $Abort_x$. Los efectos de las dos últimas acciones hacen que $setPred_x = \emptyset$. Esto implica que no es posible que el nodo que adquiere una nueva identidad simulada pertenezca al mismo tiempo al camino *wait*. Por otro lado, suponiendo que el nodo n_j recibe un mensaje INF con una nueva identidad simulada y pasa a pertenecer al camino *wait* como *candidate-known*, se observa que la identidad simulada no puede volver a a cambiar por efecto de la acción $rcvINF_{n_j}(y, m)$ ya que precisa ser *unknown* y mientras se mantenga el camino *wait* ese estado es imposible.

Cuando la acción que convierte al nodo n_j en *candidate* es $StartAddArc_{n_j}(y)$ o $rcvINF_{n_j}(y, m)$ y, además, tanto el camino p que verifica *wait* como el mensaje ALG que lo recorre se han formado antes de la ejecución de estas acciones, se pueden aplicar las mismas explicaciones que en el caso de la acción $initiate_{n_j}$ para los procesos de recepción y retransmisión del mensaje ALG entre los nodos *blocked* existentes en el camino *wait*.

■

La siguiente propiedad señala que la presencia de un nodo con su tiempo actualizado en la primera posición de una ruta, almacenada en una variable *set_st_avs* o incluida en algún mensaje *AVS* o *AVSRSP*, implica que ese nodo ya respondió al mensaje ALG de su candidato sucesor avisándole de que su identidad era mayor.

Propiedad B.2.70. Sea $p \in P^+$ tal que $first(p) = (x, t_x)$, sea $\{i, j, k\} \subseteq \mathcal{N}$, sea $t \in \mathbb{N}$, sea $sid \in T$ y sean los booleanos *fwd* y *bool*. Se verifica que $(AVSRSP, t, sid, p) \in s.channel(j, i) \vee (AVS, t, sid, p, fwd) \in s.channel(j, k) \vee (sid, t, p, bool) \in s.set\_st\_avs_k \wedge t_x = s.ta_x \Rightarrow s.nofirstAVS_x = false$.

Demostración: Se van a considerar las acciones cuyos efectos modifican las variables que intervienen en la propiedad o las que generan y/o eliminan mensajes de los tipos que en ella se citan.

- $\pi_z \in \{StartAddArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$ o $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$ o $\pi_z \in \{rcvINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Al ejecutarse estas acciones y cambiar el estado del nodo x a *candidate*, $s_z.nofirstAVS_x = true$. El valor de esta variable hace que el consecuente de la propiedad sea falso en s_z. Por tanto, la propiedad será cierta sólo si el antecedente en las distintas situaciones que llevan a este efecto es falso.

  La aplicación de la propiedad B.2.58 viene determinada por la existencia de un nodo n_i tal que, dada una ruta $path = (n_1,\ t_1)(n_2,\ t_2)\ \ldots\ (n_j,\ t_j),\ (n_i,\ t_i) \notin s_{z-1}.setPred_{n_{i+1}}$ y, o bien $i = 1$ y $s_{z-1}.blocker_{n_1} = n_2$, o bien $(n_{i-1},\ t_{i-1}) \in s_{z-1}.setPred_{n_i}$. Si no existe el nodo n_i entonces $s_{z-1}.blocker_{n_1} \neq n_2$, siendo $n_1 = x$. Para conseguir que el nodo n_1 deje de estar bloqueado por el nodo n_2 es necesario que se ejecute la acción $EndDelArc_{n_1}(n_2)$ en $z_1 <$ *z-1*. Otro de los efectos de esta acción consiste en que $s_{z_1}.ta_{n_1} < s_{z-1}.ta_{n_1}$. Como la propiedad que se está intentando demostrar exige que $t_x = s_{z-1}.ta_x$, si la ruta se mantiene intacta, $t_x = s_{z_1}.ta_x$, después de ejecutar $EndDelArc_{n_1}(n_2)$, se obtiene que $t_x < s_{z-1}.ta_x$. Por lo tanto, se ha comprobado que si no existe un nodo n_i de la ruta que cumpla las condiciones anteriores, el antecedente de la propiedad que se analiza es falso y la propiedad se verifica. Suponiendo que es aplicable la propiedad B.2.58 y existe un nodo n_i, como los efectos de estas acciones no generan los mensajes de interés, se deduce que esos mensajes ya existían en s_{z-1}. Según la propiedad B.2.58, $s_{z-1}.status.alg_x = candidate$ lo que impide que las acciones analizadas se ejecuten. La precondición para que esté habilitada la acción $StartAddArc_x(y)$ es $s_{z-1}.status.alg_x = active$, para ejecutar la acción $initiate_x$ es necesario que $s_{z-1}.status.alg_x = blocked$ y la ejecución de $rcvINF_x(y,\ m)$ será posible si $s_{z-1}.status.alg_x \in \{active,\ blocked\}$ porque $s_{z-1}.status.id_x = unknown$ (propiedad B.2.15).

  Por otra parte, la acción $rcvINF_x(y,\ m)$ puede contar entre sus efectos con la creación de mensajes *AVSRSP*. La estructura de la ruta de estos mensajes se basa en la que, a su vez, poseían los mensajes almacenados en $s_{z-1}.set\_st\_avs_x$. La hipótesis inductiva establece que estos mensajes cumplían la propiedad en s_{z-1}. En consecuencia, los mensajes que se construyen a partir de ellos también verifican la propiedad en s_z ya que no se producen cambios que afecten a los requerimientos de la propiedad.

- $\pi_z \in \{StartDelArc_n(y,\ t)\colon \{n,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Uno de los posibles efectos de la acción consiste en la formación de mensajes *AVS*, m, a partir de mensajes,

$m\prime$, almacenados en $s_{z-1}.set\_st\_avs_x$. Como $first(m.path) = first(m\prime.path)$, la propiedad se verifica por hipótesis inductiva.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Los efectos de esta acción modifican el tiempo de activación del nodo x, de manera que $s_z.ta_x > s_{z-1}.ta_x$. Resulta evidente que, al ejecutar la acción, el antecedente de la propiedad pasa a ser falso porque $t_x < s_z.ta_x$. Suponiendo que el antecedente es falso en s_{z-1}, se tiene que $t_x < s_{z-1}.ta_x$ según la propiedad B.2.1. En consecuencia, el incremento del tiempo de activación mantiene el antecedente de la propiedad como falso y la propiedad se cumple.

- $\pi_z \in \{Abort_x\colon x \in \mathcal{N}\}$. Entre los efectos de esta acción se encuentra el incremento del tiempo de activación del nodo x y el cambio a *true* de $s_z.nofirstAVS_x$. Para cualquier mensaje de los que indica la propiedad, el consecuente de la propiedad será falso. En esos casos, resulta evidente que el antecedente también es falso porque $s_{z-1}.ta_x < s_z.ta_x$. Aunque $t_x = s_{z-1}.ta_x$, el incremento del tiempo de activación hace que esa igualdad deje de cumplirse. Considerando que $t_x \neq s_{z-1}.ta_x$ (antecedente falso), la variación del tiempo de activación del nodo x mantiene la desigualdad. La propiedad B.2.1 indica que, en ese supuesto, $t_x < s_{z-1}.ta_x$. Luego, resulta evidente que $t_x < s_z.ta_x$ y el antecedente es falso como se quería demostrar.

  Cuando se ejecuta esta acción también es posible que se generen mensajes *AVS*, m. La ruta de estos nuevos mensajes difiere de las rutas almacenadas en $s_{z-1}.set\_st\_avs_x$ tan sólo en su último elemento. Esto implica que $first(m.path)$ de los mensajes *AVS* tiene las mismas características que el primer elemento de las rutas almacenadas en $s_{z-1}.set\_st\_avs_x$ que han sido utilizadas para su formación.

  Hay que recordar que, tras la ejecución de la acción, $s_z.set\_st\_avs_x = \emptyset$. Por tanto, ese efecto puede provocar que el antecedente de la propiedad sea falso en s_z.

- $\pi_z \in \{firstAVS_x\colon x \in \mathcal{N}\}$. Al ejecutar esta acción, se genera un mensaje *AVS* que verifica que $t_x = s_z.ta_x$ y $s_z.nofirstAVS_x = false$.

- $\pi_z \in \{rcvAVS_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. El el antecedente de la propiedad podría hacerse falso en s_z porque se retira del canal el mensaje m. Si este efecto de la acción viene acompañado de la formación de un nuevo mensaje *AVS*, $m\prime$, se verificará la propiedad porque se crea a partir del mensaje *AVS* recibido. Como este mensaje cumplía la propiedad en s_{z-1} por la hipótesis inductiva, sólo es necesario comprobar que $first(m.path) = first(m\prime.path)$.

Otro posible efecto de la acción es el almacenamiento en $s_z.set\_st\_avs_x$ del mensaje m. Al igual que en el efecto anterior, la hipótesis inductiva garantiza que el mensaje almacenado verifica la propiedad porque en s_{z-1} el mensaje m la cumplía y su ruta no ha cambiado al almacenarse.

Por último, el mensaje *AVS* recibido se puede transformar en un mensaje *AVSRSP*. En este caso, la propiedad también se cumple porque la ruta de m pasa a ser la ruta del nuevo mensaje *AVSRSP*. Como la hipótesis inductiva establece que el mensaje *AVS* cumplía la propiedad en s_{z-1}, el mensaje *AVSRSP* también la verifica en s_z.

- $\pi_z \in \{rcvAVSRSP_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{AVSRSP}\}$. La retirada del canal del mensaje *AVSRSP* podría hacer falso el antecedente de la propiedad. En otros casos, la hipótesis inductiva asegura que la propiedad se cumple en s_z.

- $\pi_z \in \{sndAVS_n\colon x \in \mathcal{N}\}$. Cuando se ejecuta esta acción se forma un mensaje *AVS*, m. Observando su ruta, se deduce que $first(m.path) = first(m\prime.path)$, donde $m\prime \in s_{z-1}.set\_st\_avs_n$. Por la propiedad B.2.21, se sabe que $n = last(m\prime.path)$. Así que, los cambios que pudieran afectar al nodo n no tienen ningún interés para el nodo que se analiza en la propiedad. Tras la generación del mensaje *AVS*, se elimina el mensaje $m\prime$ almacenado en $s_{z-1}.set\_st\_avs_n$. Este efecto podría hacer falso el antecedente de la propiedad en s_z.

- $\pi_z \in \{sndAVSRSP_n\colon n \subseteq \mathcal{N}\}$. Al ejecutar esta acción, se crea un nuevo mensaje *AVSRSP*, m. Según la estructura del mismo, $first(m.path) = first(m\prime.path)$, siendo $m\prime \in s_{z-1}.set\_st\_avs_n$. La hipótesis inductiva garantiza que la propiedad es cierta en s_z ya que en s_{z-1} lo era para el mensaje $m\prime$ que se utiliza en la construcción de m. A diferencia de la acción anterior, el mensaje $m\prime$ no se elimina de $s_{z-1}.set\_st\_avs_n$ y se procede a marcarlo como mensaje usado en su correspondiente campo *bool*. El cambio a *true* de este campo no influye en el cumplimiento de la propiedad.

■

La siguiente propiedad indica que si a un nodo *candidate* va dirigido un mensaje *ALG* con tiempo correcto, ese nodo no tiene información almacenada en su variable *st_alg*.

Propiedad B.2.71. Sea $\{i, j\} \subseteq \mathcal{N}$, sea $sid \in T$ y $p \in P^+$. $s.status.alg_i = candidate \wedge (ALG, s.ta_i, sid, p) \in s.channel(j, i) \Rightarrow s.st\_alg_i = NULL$.

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ por lo que la propiedad es cierta.

Hay un efecto común en las acciones $\pi_z \in \{StartAddArc_x(y)\colon \{x, y\} \subseteq \mathcal{N}\}$, $\pi_z \in \{EndAddArc_x(y, t)\colon \{x, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$, $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$ y $\pi_z \in \{rcvINF_x(y, m)\colon \{x, y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$ que consiste en la formación de un mensaje ALG con destino el nodo $id \neq x$, siendo $(id, t) \in s_z.setPred_x$. Suponiendo que $s_{z-1}.status.alg_{id} = candidate$ y $s_{z-1}.st\_alg_{id} \neq NULL$, la propiedad podría llegar a ser falsa porque, al ejecutar cualquiera de esas acciones, esas variables no se modifican. La propiedad B.2.29 asegura que, en esas condiciones, $s_z.t\_unk_{id} = s_z.ta_{id}$. Por otra lado, la propiedad B.2.46 establece que no es posible que exista un mensaje ALG hacia el nodo id con tiempo correcto. Dado que se está analizando el efecto que genera un mensaje ALG correcto, se concluye que el supuesto de partida es falso. Así que, $s_{z-1}.st\_alg_{id} = NULL$ haciendo el consecuente cierto en s_{z-1} y, por tanto, en s_z. De esta manera, queda demostrado que la propiedad también se cumple para el mensaje ALG que se pudiera formar en la ejecución de estas acciones.

A continuación se analizan la influencia de otros efectos en el cumplimiento de la propiedad:

- $\pi_z \in \{StartAddArc_x(y)\colon \{x, y\} \subseteq \mathcal{N}\}$. Los estados resultantes para el nodo x al ejecutar la acción son $s_z.status.alg_x \in \{blocked, candidate\}$. Es evidente que la propiedad se cumple cuando $s_z.status.alg_x = blocked$. Por el contrario, si $s_z.status.alg_x = candidate$, el antecedente podría llegar a ser cierto. Este estado se alcanza, siempre y cuando $s_{z-1}.status.alg_x = active$ y $s_{z-1}.status.id_x = known$. La propiedad B.2.13 establece que, en esas condiciones, $s_{z-1}.st\_alg_x = NULL$ (consecuente cierto). Los efectos que acompañan al cambio de $status.alg_x$ no modifican la variable st_alg_x. Se concluye, entonces, que la propiedad, cuando $s_z.status.alg_x = candidate$, también se verifica.

- $\pi_z \in \{EndAddArc_x(y, t)\colon \{x, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Los efectos de esta acción no provocan cambios en las variables del nodo x ni en los mensajes dirigidos a él. Aplicando la hipótesis inductiva, se concluye que la propiedad es cierta en s_z para el nodo x.

  Por otra parte, los efectos de la acción pueden formar un mensaje ALG con destino el nodo $y \neq x$, siendo $(y, t) \in s_z.setPred_x$. Suponiendo que su estado es $candidate$ y $s_{z-1}.st\_alg_y \neq NULL$, la propiedad podría llegar a ser falsa porque, al ejecutar la acción no cambian esas variables. Sin embargo, la propiedad B.2.29 asegura que, en esas condiciones, $s_z.t\_unk_y = s_z.ta_y$. y la propiedad B.2.46 establece que entonces no puede existir un mensaje que vaya al nodo y con tiempo correcto. De esta forma, se demuestra que la propiedad cumple para el mensaje ALG que pudiera ser formado en la ejecución de esta acción.

- $\pi_z = \{StartDelArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Si la acción está habilitada porque $s_{z-1}.status.alg_x = active \neq candidate$, el antecedente de la propiedad es falso en s_{z-1}. Los efectos de la acción mantienen ese estado para el nodo x y, por tanto, la propiedad sigue siendo cierta en s_z. En el caso de que la acción esté habilitada cuando $s_{z-1}.status.alg_y = aborted$, la hipótesis inductiva concluye.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$ o $\pi_z \in \{Abort_x\colon x \in \mathcal{N}\}$ o $\pi_z \in \{rcvAVS_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Tras la ejecución de estas acciones, $s_z.status.alg_x \neq candidate$. De acuerdo con el estado alcanzado, la propiedad se cumple en s_z con el antecedente falso.

- $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$. La acción esté habilitada si $s_{z-1}.status.alg_x = active$ y $s_{z-1}.status.id_x = known$. La propiedad B.2.13 establece que, en esas condiciones, $s_{z-1}.st\_alg_x = NULL$. Como los efectos de la acción no modifican la variable st_alg_x, se concluye que la propiedad se verifica en s_z cuando $x = i$.

- $\pi_z \in \{rcvALG_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{ALG}\}$. Al retirar el mensaje ALG del canal, el antecedente de la propiedad se convierte en falso. Si el estado final del nodo x tras la ejecución de la acción es *dummy* o *victim*, la propiedad se cumple con el antecedente falso. Sin embargo, si el estado del nodo x en s_z puede ser *candidate*, los efectos de esta acción hacen que $s_z.t\_unk_x = s_z.ta_x$. La propiedad B.2.46 establece que, entonces, no existe otro mensaje con tiempo correcto dirigido al nodo x (antecedente falso).

- $\pi_z \in \{firstAVS_x\colon x \in \mathcal{N}\}$. De acuerdo con la precondición de esta acción $s_{z-1}.st\_alg_x \neq NULL$. Eso significa que la propiedad se verifica en s_{z-1} con el antecedente y el consecuente falsos. Cuando se ejecuta la acción, los efectos de la acción no influyen en las variables ni en el mensaje de la propiedad, cumpliéndose la propiedad en s_z.

- $\pi_z \in \{rcvINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Al ejecutar esta acción, $s_z.st\_alg_x = NULL$. Esto hace que el consecuente de la propiedad sea cierto en s_z para $x = i$.

- $\pi_z \in \{sndAVS_x\colon x \in \mathcal{N}\}$ o $\pi_z \in \{sndAVSRSP_x\colon x \in \mathcal{N}\}$. Una de las precondiciones de estas acciones es que $s_{z-1}.cand\_succ_x \neq \epsilon$. Según la propiedad B.2.17, el estado que le corresponde al nodo x en esa situación es *candidate*. Como $s_{z-1}.st\_avsrsp_x \neq NULL$, se tiene que $s_{z-1}.st\_alg_x \neq NULL$. Esto implica que la propiedad en s_{z-1} se cumple con el antecedente falso. Los efectos de estas acciones no modifican las variables del antecedente y tampoco generan y eliminan mensajes ALG. Por todo ello, el antecedente de la propiedad sigue siendo falso en s_z.

■

La siguiente propiedad permite afirmar que, cuando un nodo candidato recibe un mensaje ALG procedente de su candidato sucesor de identidad simulada superior a la suya, guarda adecuadamente ese mensaje en su variable *st_avsrsp*.

Propiedad B.2.72. Sea $\{n_1, n_n\} \subseteq \mathcal{N}$. Sea $p = (n_1, t_1)\ldots(n_n, t_n) \in P^+$. $\exists\ z$ tal que $\forall\ z\prime > z$, $wait(n_1, n_n, p, s_{z\prime}) \wedge \exists\ n_i, n_j \in nodes(p)$ con $i < j$, $s_{z\prime}.status.alg_{n_i} = s_{z\prime}.status.alg_{n_j} = candidate \wedge (\forall\ n_k \in nodes(p)$: $i < k < j$, $s_{z\prime}.status.alg_{n_k} \neq \{candidate, victim\}) \wedge s_{z\prime}.sim\_id_{n_j} > s_{z\prime}.sim\_id_{n_i} \wedge s_{z\prime}.nofirstAVS_{n_i} = true \Rightarrow \exists\ z\prime\prime \geq z$: $s_{z\prime\prime}.st\_avsrsp_{n_i} = (s_{z\prime\prime}.ta_{n_i}, (x, t, nu), (n_i, s_{z\prime\prime}.ta_{n_i}).s_{z\prime\prime}.st\_alg_{n_i}.path)$, siendo $x = s_{z\prime\prime}.sim\_id_{n_j}.id \wedge t = s_{z\prime\prime}.sim\_id_{n_j}.ta \wedge nu \in \mathbb{N}^*$.

Demostración: Para demostrar esta propiedad hay que estudiar dos supuestos distintos:$s_{z\prime}.st\_alg_{n_i} = NULL$ y $s_{z\prime}.st\_alg_{n_i} \neq NULL$.

- Si $s_{z\prime}.st\_alg_{n_i} = NULL$, la propiedad B.2.69 asegura que existe un mensaje ALG dirigido al nodo n_i y éste tiene la acción $rcvALG_{n_i}(n_k, m)$ habilitada. Es necesario comprobar que ninguna acción del algoritmo puede deshabilitar la acción $rcvALG_{n_i}(n_k, m)$, siendo $status.alg_{n_i} = candidate$ y $st\_alg_{n_i} = NULL$:

  - Por una parte, para ejecutar las acciones$StartAddArc_{n_i}(y)$, $Abort_{n_i}$, $initiate_{n_i}$ y $rcvINF_{n_i}(y, m)$ es preciso que $status.alg_{n_i} \neq candidate$. Las acciones $firstAVS_{n_i}$, $sndAVS_{n_i}$, $sndAVSRSP_{n_i}$ están habilitadas al ser $status.alg_{n_i} = candidate$, pero en todos esos casos es necesario $st\_alg_{n_i} \neq NULL$ y se está estudiando el caso contrario.
  - Si se ejecuta $EndDelArc_{n_i}(y)$ significa que previamente se ha roto el camino de esperas, esto es no se cumple la condición *wait*.
  - Con la acción $rcvAVS_{n_i}(y, m)$, el nodo n_i puede llegar a ser *victim*, pero no está permitido, o no cambia el estado del nodo.
  - Para el resto de acciones, como no se modifica la variable $status.alg_{n_i}$, se tiene que es posible todavía ejecutar $rcvALG_{n_i}(y, m)$.

  Por lo tanto la condición de equidad débil sobre la partición indica que la acción se ejecuta.Uno de los efectos de esta acción, dado que $s_{z\prime}.sim\_id_{n_j} > s_{z\prime}.sim\_id_{n_i}$, consiste en guardar el mensaje ALG en $s_{z\prime\prime}.st\_avsrsp_{n_i}$. Al almacenarse en $s_{z\prime\prime}.st\_avsrsp_{n_i}$ se guardan estos campos como $(s_{z\prime\prime}.ta_{n_i}, (x, t, nu), (n_i, s_{z\prime\prime}.ta_{n_i}).m.path)$, con $x = s_{z\prime\prime}.sim\_id_{n_j}.id \wedge t = s_{z\prime\prime}.sim\_id_{n_j}.ta$. Cuando se produce este efecto no ha variado la relación entre las identidades simuladas de los nodos *candidate* porque ninguna de las acciones que modifican la identidad simulada de los nodos podría estar habilitada para un nodo *candidate* de un camino *wait* de los que señala la propiedad.

- En el otro supuesto en el que debe verificarse la propiedad, $s_{z\prime}.st\_alg_{n_i} \neq NULL$, se tiene que asegurar que el mensaje almacenado se corresponde con el mensaje ALG que se dirigía al nodo n_i cuando su $st\_alg_{n_i}$ estaba vacío (condiciones de la propiedad B.2.69). Como los efectos de la recepción de ese mensaje ALG fueron los que también almacenaron el mensaje en $s_{z\prime\prime}.st\_avsrsp_{n_i}$, quedará comprobada la propiedad.

  Antes de avanzar en la demostración, es necesario asegurar que todo nodo *candidate* que recibe un mensaje ALG con tiempo correcto lo hace siempre con su st_alg vacío. De esta forma, se cumplirán todos los requisitos de la propiedad B.2.69 y se podrá aplicar sin excepciones. La propiedad B.2.71 así lo confirma.

En resumen, sea cual sea la manera en la que el nodo n_i alcanza el estado *candidate*, ya sea formando parte del camino *wait* antes o después de ello, el nodo n_i mantendrá vacío su st_alg hasta recibir un mensaje ALG. Dicho de otra forma, para que $s_{z\prime}.st\_alg_{n_i} \neq NULL$ tuvo que tener previamente habilitada la acción $rcvALG_{n_i}(y,\ m)$ siendo *candidate*. Ese mensaje ALG, dada la relación entre las identidades simuladas del nodo n_i y n_j, se almacenará convenientemente tanto en $s_{z\prime}.st\_alg_{n_i}$ como en $s_{z\prime}.st\_avsrsp_{n_i}$. El contenido de ambos almacenes podría cambiar si llegara un nuevo mensaje ALG. Por otra parte, si se recibiera un mensaje $AVSRSP$ sólo cambiaría $s_{z\prime}.st\_avsrsp_{n_i}$. A continuación, se comprobará que la recepción de un nuevo mensaje ALG o de un mensaje $AVSRSP$ son imposibles.

En primer lugar, si el camino *wait* se mantiene después de haber sido recorrido por un mensaje ALG, los nodos que lo conforman no podrán dejar pasar otro mensaje ALG.

Para descartar la existencia de un mensaje $AVSRSP$ dirigido al nodo n_i hay que recurrir a la propiedad B.2.70. Esta propiedad establece que, si el mensaje cuenta con el tiempo correcto de activación del nodo n_i, para poder ser almacenado en $s_{z\prime}.st\_avsrsp_{n_i}$, entonces la variable $s_{z\prime}.nofirstAVS_{first(path).id} = false$. Considerando la propiedad B.2.18, se deduce que $first(path).id = n_i$. El antecedente de la propiedad que se está demostrando exige que $s_{z\prime}.nofirstAVS_{n_i} = true$. En conclusión, el contenido de $s_{z\prime\prime}.st\_avsrsp_{n_i}$ hace referencia al mensaje ALG que quedó registrado en $s_{z\prime}.st\_alg_{n_i}$ y no se puede modificar mientras el camino *wait* se mantenga. Sus campos son:

- $m_{ALG}.ta = s_{z\prime\prime}.ta_{n_i} = s_{z\prime\prime}.st\_avsrsp_{n_i}.ta$ Si el mensaje se almacenó en $st\_alg_{n_i}$ quiere decir que el tiempo del mensaje coincidía con tiempo de activación del nodo n_i. Como el nodo n_i pertenece en todo momento a un camino *wait*, su tiempo de activación no ha podido variar y es igual al que tenía antes de recibir el mensaje ALG.

- $s_{z\prime\prime}.st\_alg_{n_i}.sid = s_{z\prime\prime}.st\_avsrsp_{n_i}.sid$. Considerando la propiedad B.2.55, resulta que n_j es el primer nodo almacenado de la ruta que es distinto de *dummy*. Como

la ruta de $s_{z''}.st\_alg_{n_i}.path = \gamma_1 \cdot (n_j,\ t_j) \cdot \gamma_2$, se concluye que la ruta formada por el nodo n_i concatenado con $\gamma_1 \cdot (n_j,\ t_j)$ coincide con los nodos que aparecen en el camino *wait* de la propiedad (al tratarse de un sistema de único recurso, se cumple la propiedad B.2.3). Además, se puede afirmar, o bien γ_2 es una cadena vacía, o son esperas que no existen y forman parte de un camino *ewait* porque γ_2 no se corresponde con los nodos del camino *wait*.

Por otro lado, la propiedad B.2.56 indica que $s_{z''}.sim\_id_{n_j} \geq s_{z''}.st\_alg_{n_i}.sid$ cuando $(n_j,\ t_j) \notin s_{z''}.setPred_{first(\gamma^2).id}$, siendo $s_{z''}.st\_alg_{n_i} = \gamma_1 \cdot (n_j,\ t_j) \cdot \gamma_2$. Para validar el hecho de que $(n_j,\ t_j) \notin s_{z''}.setPred_{first(\gamma^2).id}$, basta con suponer que n_j pertenece al conjunto de predecesores de $first(\gamma_2).id$ y aplicar la propiedad B.2.55 para llegar a una contradicción. Según esa propiedad, el nodo n_j debería ser *dummy* y no *candidate*, tal y como impone la propiedad que se trata de demostrar.

En consecuencia, se confirma que $s_{z''}.sim\_id_{n_j}.nu \geq s_{z''}.st\_avsrsp_{n_i}.sid.nu$ porque $s_{z''}.sim\_id_{n_j}.id = s_{z''}.st\_alg_{n_i}.sid.id \wedge s_{z''}.sim\_id_{n_j}.ta = s_{z''}.st\_alg_{n_i}.sid.ta$.

- $(n_i,\ s_{z''}.ta_{n_i}).s_{z''}.st\_alg_{n_i}.path = s_{z''}.st\_avsrsp_{n_i}.path$. Teniendo en cuenta la propiedad B.2.70 y los efectos de la acción $rcvALG_{n_i}(n,\ m)$, es evidente que la estructura de la ruta guardada en $s_{z''}.st\_avsrsp_{n_i}$ cumple el consecuente de la propiedad.

■

En esta propiedad se garantiza la generación de un mensaje *AVS* cuando un nodo candidato recibe un mensaje *ALG* de un candidato sucesor de identidad simulada inferior.

Propiedad B.2.73. Se verifica que $\exists\ z$ tal que $\forall\ z\prime > z\ wait(n_1,\ n_n,\ p,\ s_{z\prime}) \wedge \exists\ n_i,\ n_j \in nodes(p)$ con $i < j \wedge s_{z\prime}.status.alg_{n_i} = s_{z\prime}.status.alg_{n_j} = candidate \wedge (\forall\ n_k \in nodes(p)$: $i < k < j$: $s_{z\prime}.status.alg_{n_k} \neq \{candidate,\ victim\}) \wedge s_{z\prime}.sim\_id_{n_j} < s_{z\prime}.sim\_id_{n_i} \Rightarrow \exists\ z\prime\prime \geq z$: $\exists\ m$: $(AVS,\ s_{z''}.sim\_id_{n_i},\ last(s_{z''}.st\_alg_{n_i}.path).ta,\ (n_i,\ s_{z''}.ta_{n_i}) \cdot s_{z''}.st\_alg_{n_i}.path,\ false) \in s_z.channel(n_i,\ last(s_{z''}.st\_alg_{n_i}.path).id))$.

Demostración: Supóngase, en primer lugar, que $s_{z\prime}.st\_alg_{n_i} = NULL$. Al aplicar la propiedad B.2.69, se asegura la existencia de un mensaje *ALG* con destino el nodo n_i que fue reenviado por el nodo n_j y quedará habilitada la acción $firstAVS_{n_i}$ porque sus precondiciones se cumplen:

- $s_{z\prime}.status.id_{n_i} = known$ obvio porque $s_{z\prime}.status.alg_{n_i} = candidate$ (propiedad B.2.15)

- $s_{z\prime}.st\_alg_{n_i} \neq NULL$, efecto de la acción $rcvALG_{n_i}(n_k,\ m)$
- $s_{z\prime}.cand\_succ_{n_i} \neq NULL$, efecto de la acción $rcvALG_{n_i}(n_k,\ m)$
- $nofirstAVS_i = true$, se exige en el antecedente de la propiedad. Tras la recepción del mensaje ALG, esta variable mantiene su valor.
- $s_{z\prime}.cand\_succ_{n_i} < s_{z\prime}.sim\_id_{n_i}$, $s_{z\prime}.cand\_succ_{n_i} = m_{ALG}.sid = s_{z\prime}.st\_alg_{n_i}.sid$. Teniendo en cuentas los posibles valores que puede contener $s_{z\prime}.cand\_succ_{n_i}$ se tiene que:
  - $cand\_succ_{n_i} = s_{z\prime}.sim\_id_{n_j}$, el nodo n_j ejecuta $initiate_{n_j}$ y el mensaje ALG llega hasta el nodo n_i ($s_{z\prime}.sim\_id_{n_j} = (n_j,\ s_{z\prime}.ta_{n_j},\ \epsilon)$).
  - el nodo n_j siendo *dummy* reenvía el mensaje ALG que llega hasta el nodo n_i y después se convierte en *candidate*: $s_{z\prime}.cand\_succ_{n_i} < s_{z\prime}.sim\_id_{n_j}$ ($s_{z\prime}.cand\_succ_{n_i}.id = s_{z\prime}.sim\_id_{n_j}.id$, $s_{z\prime}.cand\_succ_{n_i}.ta = s_{z\prime}.sim\_id_{n_j}.ta$ pero $s_{z\prime}.cand\_succ_{n_i}.nu < s_{z\prime}.sim\_id_{n_j}.nu$).
  - el nodo n_j adquiere una nueva identidad simulada, pasa a *candidate* y la envía en el mensaje ALG que recibe n_i: $s_{z\prime}.cand\_succ_{n_i} = s_{z\prime}.sim\_id_{n_j}$.

  En el antecedente de la propiedad se impone que $s_{z\prime}.sim\_id_{n_j} < s_{z\prime}.sim\_id_{n_i}$. Según las relaciones anteriores, $cand\_succ_{n_i} \leq sim\_id_{n_j}$. De esta manera queda comprobado que $cand\_succ_{n_i} < sim\_id_{n_i}$.

Finalmente, dado que como la acción $firstAVS_{n_i}$ se ejecutará por la propiedad de equidad débil sobre la partición, se comprueba que ninguna de las acciones del algoritmo puede deshabilitarla:

- Para ejecutar las acciones $StartAddArc_{n_i}(y)$, $Abort_{n_i}$, $initiate_{n_i}$ y $rcvINF_{n_i}(y,\ m)$ es preciso que $status.alg_{n_i} \neq candidate$. Si se ejecuta $StartDelArc_{n_i}(y,\ t)$, o bien $status.alg_{n_i} \neq candidate$, o bien $status.alg_y = aborted$ e $y \notin nodes(p)$. En este último caso, las variables que componen las precondiciones de $firstAVS_{n_i}$ no cambian.
- Si se ejecuta $EndDelArc_{n_i}(y)$ significa que previamente se ha roto el camino de esperas y no se cumple la propiedad *wait*.
- $rcvALG_{n_i}(y,\ m)$. El nodo n_i no puede recibir otro mensaje ALG con tiempo correcto a no ser que previamente su $st\_alg_{n_i} = NULL$ (propiedad B.2.71).
- La acción $rcvAVS_{n_i}(y,\ m)$ puede estar habilitada, pero no cambia las variables de la precondición. Como la propiedad *wait* se mantiene, el nodo n_i permanecerá siendo *candidate*. El antecedente de la propiedad no permite el efecto que lo convierte en *victim*.

- La acción $rcvAVSRSP_{n_i}(y,\ m)$ cambia el valor de $cand\_succ_{n_i}$ si el mensaje es admitido (recoge el tiempo actual del nodo x). En ese caso, la propiedad B.2.70 señala que la acción $s_{z\prime}.nofirstAVS_{n_i} = false$ y el antecedente de la propiedad indica que $s_{z\prime}.nofirstAVS_{n_i} = true$ en la recepción del mensaje ALG.

- Las acciones $sndAVS_{n_i}$ y $sndAVSRSP_{n_i}$ están habilitadas si $status.alg_{n_i} = candidate$. Además, en todos esos casos, $st\_avsrsp_{n_i} \neq NULL$. Dadas las condiciones de habilitación de la acción $firstAVS_{n_i}$, la propiedad B.2.51 establece que $st\_avsrsp_x = NULL$. En consecuencia, las acciones $sndAVS_{n_i}$ y $sndAVSRSP_{n_i}$ no pueden ser ejecutadas.

- Para el resto de acciones, como no modifican las variables que incluye la precondición de la acción $firstAVS_{n_i}$, la acción está todavía habilitada.

Seguidamente, se va a considerar que $s_{z\prime}.st\_alg_{n_i} \neq NULL$. De acuerdo con la propiedad B.2.72 se concluye que el mensaje ALG que se dirigía al nodo n_i es precisamente el que se ha almacenado. Con la relación impuesta a las identidades simuladas del nodo n_i y n_j, también quedará habilitada la acción $firstAVS_{n_i}$.

El contenido del almacén $st\_alg_{n_i}$ podría cambiar si llegara un nuevo mensaje ALG. Sin embargo, se comprobará que la recepción de un nuevo mensaje ALG es imposible. Si el camino *wait* se mantiene después de haber sido recorrido por un mensaje ALG, los nodos que lo conforman no podrán dejar pasar un nuevo mensaje ALG. Ya ha quedado demostrado que el nodo *candidate* que recibe un mensaje ALG debe presentar su $st\_alg_{n_i} = NULL$ (propiedad B.2.71). Así que, en el supuesto del análisis, $s_{z\prime}.st\_alg_{n_i} \neq NULL$, el nodo n_i no puede recibir un nuevo mensaje ALG. En caso de hacerlo, $s_{z\prime}.st\_alg_{n_i} = NULL$ y se aplica directamente la propiedad B.2.69.

El efecto de la acción $firstAVS_{n_i}$ consiste en la creación de un mensaje de tipo AVS. Este mensaje se construye a partir de $s_{z\prime\prime}.st\_alg_{n_i}$ y tiene la siguiente estructura: $(AVS,\ s_{z\prime\prime}.sim\_id_{n_i},\ last(s_{z\prime\prime}.st\_alg_{n_i}.path).ta,\ (n_i,\ s_{z\prime\prime}.ta_{n_i}).s_{z\prime\prime}.st\_alg_{n_i}.path,\ false) \in s_z.channel(n_i,\ last(s_{z\prime\prime}.st\_alg_{n_i}.path).id)$.

■

Esta propiedad indica que todo nodo candidato que envió un mensaje AVS directo a su candidato predecesor tiene almacenada la ruta que les separa en su correspondiente st_alg.

Propiedad B.2.74. $s.nofirstAVS_i = false \wedge s.status.alg_i = candidate \Rightarrow s.st\_alg_i \neq NULL$

Demostración: En el estado inicial, s_0, $\forall\ i \in \mathcal{N}$ se cumple que $s_0.status.alg_i = active$ por lo que la propiedad es cierta. A continuación, se analizan las acciones que modifican las variables de la propiedad.

- $\pi_z \in \{StartAddArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$. Al ejecutarse la acción, $s_z.status.alg_x = \{blocked,\ candidate\}$. En el primer caso, es evidente que el antecedente de la propiedad es falso. En el segundo caso, el cambio de estado conlleva que $s_z.nofirstAVS_x = true$. Por lo tanto, en esta situación, la propiedad también se cumple con el antecedente falso.

- $\pi_z \in \{StartDelArc_x(y,\ t)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Si la acción se ejecuta siendo $s_{z-1}.status.alg_x = active$, esta variable conserva su valor y la propiedad queda demostrada porque el antecedente es falso. Cuando $s_{z-1}.status.alg_y = aborted$ y la acción se ejecuta, la hipótesis inductiva concluye porque los efectos, en este supuesto, no cambian ninguna de la variables de la propiedad.

- $\pi_z \in \{EndDelArc_x(y)\colon \{x,\ y\} \subseteq \mathcal{N}\}$ o $\pi_z \in \{Abort_x\colon x \in \mathcal{N}\}$. Los efectos de estas acciones hacen que $s_z.status.alg_x \neq candidate$. Como el antecedente es falso en s_z, la propiedad se cumple.

- $\pi_z \in \{initiate_x\colon x \in \mathcal{N}\}$. Los efectos de la acción provocan que $s_z.status.alg_x = candidate$ pero $s_z.nofirstAVS_x = true$. Esto implica que la propiedad se cumple con el antecedente falso.

- $\pi_z \in \{rcvALG_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{ALG}\}$. Cuando el mensaje ALG que se retira del canal tiene el tiempo correcto, es obvio que $s_z.st\_alg_x \neq NULL$. Sin embargo, cuando el mensaje ALG tiene un tiempo incorrecto, es la hipótesis inductiva la que concluye.

- $\pi_z \in \{firstAVS_x\colon x \in \mathcal{N}\}$. Una de las condiciones de habilitación de la acción indica que $s_{z-1}.st\_alg_x \neq NULL$. Como los efectos de la acción no cambian esta variable, la propiedad se cumple con el consecuente cierto.

- $\pi_z \in \{rcvINF_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{INF}\}$. Tras ejecutarse esta acción, $s_z.status.alg_x = \{blocked,\ active,\ candidate\}$. En los dos primeros casos, la propiedad se cumple con el antecedente falso. Si el nodo x se convierte en *candidate*, al mismo tiempo, la variable $s_z.nofirstAVS_x$ adquiere el valor *true*. Esto quiere decir que el antecedente de la propiedad en s_z es falso y, en consecuencia, la propiedad se cumple.

- $\pi_z \in \{rcvAVS_x(y,\ m)\colon \{x,\ y\} \subseteq \mathcal{N} \wedge m \in M_{AVS}\}$. Si los efectos de la acción convierten $s_z.status.alg_x$ en *victim*, es evidente que el antecedente de la propiedad es falso. Considerando el resto de posibles efectos de la propiedad, la hipótesis inductiva permite demostrar que la propiedad se verifica.

■

La siguiente propiedad permite asegurar que, en una ruta válida encabezada y terminada por nodos candidatos, se dispone de información suficiente del nodo de mayor identidad simulada. Si el nodo final de esa ruta es el de mayor identidad simulada, el nodo inicial de la ruta, n_1, conoce su identidad simulada y la tiene almacenada en su correspondiente *cand_succ*. Por el contrario, si el primer nodo de la ruta es el de mayor identidad simulada, el nodo final n_j tiene registrada la identidad simulada de su candidato predecesor no inmediato como parte de un mensaje almacenado en $set\_st\_avs_{n_j}$.

Propiedad B.2.75. Sea $\alpha = s_0\ \pi_1\ s_1 \ldots \pi_z\ s_z \ldots$ una ejecución de S. Sea $p \in P^+$ tal que $p = (n_1,\ t_1).(n_2,\ t_2)\ldots(n_j,\ t_j)$. Se verifica que $\exists\ z_0$: $\forall\ z \geq z_0$ tal que $wait(n_1,\ n_j,\ p,\ s_z) \wedge s_z.status.alg_{n_1} = s_z.status.alg_{n_j} = candidate \wedge \forall\ n \in nodes(p),\ s_z.status.alg_n \neq victim$:

- $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1} > max(\{s_z.sim\_id_{n_l}$: $1 < l < j,\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\}) \Rightarrow \exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se verifica que $s_{z_n}.cand\_succ_{n_1} = (s_{z_n}.sim\_id_{n_j}.id,\ s_{z_n}.sim\_id_{n_j}.ta,\ nu)$,siendo $nu \leq s_{z_n}.sim\_id_{n_j}.nu \wedge$

- $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_j} > max(\{s_z.sim\_id_{n_l}$: $1 < l < j,\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\}) \Rightarrow \exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se verifica que $(s_{z_n}.sim\_id_{n_1},\ t,\ p,\ false) \in s_{z_n}.set\_st\_avs_{n_j}$ tal que $t_1 = s_{z_n}.ta_{n_1} \wedge t \in \mathbb{N}$.

Demostración: Esta propiedad se va a demostrar por inducción sobre el número de nodos en estado *candidate* de ruta p que cumple la definicón de *wait*. En lo que sigue se llamará L a esta cantidad, esto es, $L = ||\{n_l \in nodes(p)$: $s_z.status\_alg_{n_l} = candidate\}||$, siendo $1 \leq l \leq j$.

Caso base de inducción $L = 2$:

En el caso inicial de la inducción se considerará que sólo hay dos nodos candidatos en la ruta p que coinciden con el primero y el último de sus nodos, n_1 y n_j. Por tanto, se asume que no hay ningún nodo intermedio en la ruta p que sea *candidate* o *victim*. Además, la parte del antecedente de la propiedad que alude a las identidades simuladas de los nodos candidatos queda reducida a: $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$ y $s_z.sim\_id_{n_j} < s_z.sim\_id_{n_1}$. A continuación, se analizan estas dos posibilidades para el caso base de la inducción.

$s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$:

Según la manera en la que el nodo n_j pasó a ser *candidate* se plantean diversas situaciones:

Opción 1:

Por efecto de la acción $initiate_{n_j}$, el nodo n_j se convirtió en *candidate* y mandó un mensaje ALG a n_1. En esta situación, se asume que la ruta almacenada en $s_z.st\_alg_{n_1}$ se corresponde con los nodos de p. Esto implica que el mensaje ALG cumplía los requisitos de la propiedad B.2.69. Por otra parte, la propiedad B.2.71 confirma que, antes de la recepción del mensaje ALG y de su correspondiente almacenamiento en $s_z.st\_alg_{n_1}$, $st\_alg_{n_1} = NULL$. Además, la propiedad B.2.74 asegura que $nofirstAVS_{n_1} = true$ ya que $status.alg_{n_1} = candidate$ y $st\_alg_{n_1} = NULL$. Por todo ello, en esta opción no hay posibilidad de que el nodo n_1 ejecute la acción $firstAVS_{n_1}$ dado que en este supuesto $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$, o lo que es lo mismo, $s_z.cand\_succ_{n_1} > s_z.sim\_id_{n_1}$. Así que la propiedad B.2.72 es aplicable y se demuestra que la identidad simulada del nodo sucesor se corresponde con la que indica la propiedad.

Opción 2:

Un nodo *candidate* ajeno al camino p mandó el mensaje ALG y la ruta almacenada en $s_z.st\_alg_{n_1}$ se corresponde con los nodos de p seguidos de $\gamma_2 \neq \epsilon$. En esta situación, el nodo n_j quedó en estado *dummy* al dejar pasar el mensaje ALG y para convertirse en *candidate* tuvo que activarse. Que el nodo n_j se activara, supone que las esperas que lo unían con el nodo emisor del mensaje ALG también se eliminaron. Además de cambiar su estado a *candidate*, el nodo n_j también adopta una nueva identidad simulada que se corresponde con la del nodo que envió el mensaje ALG o con la de otro nodo *candidate* cuyo mensaje ALG llegó hasta ese nodo emisor. Cuando el nodo n_1 recibe el mensaje ALG se verifica tanto la propiedad B.2.71 como la propiedad B.2.69. En consecuencia, si, antes de recibir el mensaje ALG $st\_alg_{n_1} = NULL$ y $status.alg_{n_1} = candidate$, la propiedad B.2.74 establece que $nofirstAVS_{n_1} = true$. Al almacenarse el mensaje ALG en $s_z.st\_avsrsp_{n_1}$ como indica la propiedad B.2.72, la variable $nofirstAVS_{n_1}$ mantiene su valor. Según esto, el nodo n_1 no ha tenido la oportunidad de ejecutar la acción $firstAVS_{n_1}$. A partir del campo *sid* de $s_z.st\_avsrsp_{n_1}$ se comprueba que la identidad del candidato sucesor del nodo n_1 coincide con la adquirida por el nodo n_j tal y como señala la propiedad.

Opción 3:

Si el nodo n_1 no formaba parte inicialmente de la ruta p, en s_{z_1}, siendo $z_1 < z$, el nodo n_1 podía cumplir o no las condiciones de habilitación de la acción $firstAVS_{n_1}$. Así que, seguidamente se estudian las dos posibilidades.

1. $s_{z_1}.nofirstAVS_{n_1} = true$ (no se ejecutó la acción $firstAVS_{n_1}$):

En esta situación, tanto $s_{z_1}.st\_alg_{n_1}$ como $s_{z_1}.st\_avsrsp_{n_1}$, almacenaron una ruta completamente distinta a p y todas las esperas de la ruta guardada en $s_{z_1}.st\_alg_{n_1}$ deben desaparecer para que pueda surgir p en s_z. Para ello, se ejecutó en s_{z_2}, la acción $EndDelArc_{n_1}(n)$ $(z > z_2 > z_1)$ que sea cual sea el valor de $s_{z_2-1}.inf\_need_{n_1}$

tiene como efecto que $s_{z_2}.st\_avsrsp_{n_1} = NULL$. El resto del análisis se centra en el valor que adquiere $s_{z_2}.status.id_{n_1}$:

- $s_{z_2}.status.id_{n_1} = known$. Al ejecutar la acción $StartAddArc_{n_1}(n_2)$ que permite construir la espera que marca el camino *wait* en s_z, el nodo n_1 puede convertirse en *candidate* directamente como efecto de esta acción o tener que ejecutar posteriormente la acción $initiate_{n_1}$. En cualquiera de estas evoluciones se mantiene $nofirstAVS_{n_1} = true$ y se podrá aplicar la propiedad B.2.72 igual que sucedía en la primera y en la segunda opción.

- $s_{z_2}.status.id_{n_1} = unknown$. Con este estado, la acción $StartAddArc_{n_1}(n_2)$ puede ejecutarse antes o después de la acción $rcvINF_{n_1}(n)$ que transforma $status.id_{n_1} = known$. Al igual que en el apartado anterior, el nodo n_1 se convertirá de forma directa en *candidate* o precisará posteriormente la ejecución de la acción $initiate_{n_1}$.

2. $s_{z_1}.nofirstAVS_{n_1} = false$ (se ejecutó la acción $firstAVS_{n_1}$):

Como efecto de la acción, se generó un mensaje *AVS* que tenía como destino un nodo n_l que también era *candidate*. La precondición de esta acción indicaba que $s_{z_1}.sim\_id_{n_1} > s_{z_1}.sim\_id_{n_l}$. Sabiendo que $s_{z_1}.st\_alg_{n_1}$ almacenaba una ruta completamente distinta a p, es necesario que todas las esperas registradas en la ruta de $s_{z_1}.st\_alg_{n_1}$ desaparecieran para que surgiera p en s_z. Para ello, se ejecutó en s_{z_2}, la acción $EndDelArc_{n_1}(n)$ $(z > z_2 > z_1)$. Según los efectos de esta acción para un nodo *candidate*, pueden darse diferentes evoluciones.

- $s_{z_2}.status.alg_{n_1} = active \wedge s_{z_2}.status.id_{n_1} = known \wedge s_{z_2}.st\_alg_{n_1} = NULL$.

  En este caso, el nodo n_1 ejecutará $StartAddArc_{n_1}(n_2)$ en s_{z_3} $(z > z_3 > z_2)$ con el objeto de formar la ruta p. Si $s_{z_3-1}.setPredToInf_{n_1} \neq \emptyset$, resultará que $s_{z_3}.status.alg_{n_1} = candidate$ y $s_{z_3}.nofirstAVS_{n_1} = true$. En estas condiciones, el nodo n_1 recibirá un mensaje *ALG* del nodo n_j que es *candidate* y cumple la propiedad B.2.69. Por otra parte, la existencia de la ruta p cumpliendo la definición de $wait(n_1, n_j, p, s_z)$ junto con la relación $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$ confirma que se puede utilizar la propiedad B.2.72 para demostrar que se obtiene la información del candidato sucesor de n_1.

  Suponiendo que $s_{z_3-1}.setPredToInf_{n_1} = \emptyset$, los efectos de la $StartAddArc_{n_1}(n_2)$ hacen que $s_{z_3}.status.alg_{n_1} = blocked$ y el resto de variables mantienen su valor. Como $wait(n_1, n_j, p, s_z)$ requiere que $s_z.status.alg_{n_1} = candidate$, el nodo n_1 sólo podrá ejecutar la acción $initiate_{n_1}$ $(setPred_{n_1} \neq \emptyset)$ para conseguir ese cambio de estado y alcanzar las condiciones marcadas en la propiedad que se analiza. Al adquirir ese nuevo estado, $s_z.nofirstAVS_{n_1} = true$ y se puede concluir que,

finalmente, el nodo n_1 guardará en $s_z.st\_avsrsp_{n_1}$ la información de interés tal y como indica la propiedad B.2.72.

- $s_{z_2}.status.alg_{n_1} = active \wedge s_{z_2}.status.id_{n_1} = unknown \wedge s_{z_2}.st\_alg_{n_1} \neq NULL$.

  Considerando este caso, el nodo n_1 podrá ejecutar la acción $rcvINF_{n_1}(n,\ m)$ en s_{z_3} $(z > z_3 > z_2)$, antes o después de bloquearse por el nodo n_2 que le sucederá en la ruta p. Si $s_{z_3}.setPredToInf_{n_1} \neq \emptyset$ y $s_{z_3}.status.alg_{n_1} = blocked$, los efectos de esta acción permitirán alcanzar las condiciones que marcan las propiedades B.2.69 y B.2.72, esto es, $s_{z_3}.status.alg_{n_1} = candidate$, $s_{z_3}.st\_alg_{n_1} = NULL$ y $s_{z_3}.nofirstAVS_{n_1} = true$. Sin embargo, cuando $s_{z_3}.setPredToInf_{n_1} \neq \emptyset$ y $s_{z_3}.status.alg_{n_1} = active$ (el nodo n_1 no se ha bloqueado aún por el nodo que le sigue en p), se observarán las siguientes características: $s_{z_3}.status.alg_{n_1} = active$, $s_{z_3}.status.id_{n_1} = known$, $s_{z_3}.st\_alg_{n_1} = NULL$ pero $s_{z_3}.nofirstAVS_{n_1} = false$. Al bloquearse n_1 por el nodo n_2, se conseguirá que $s_z.nofirstAVS_{n_1} = true$ tal y como se precisa para aplicar la propiedad B.2.72. El nodo n_1 finalmente recibirá un mensaje ALG de n_j, siendo *candidate*.

  Por último, hay que estudiar en este caso la evolución del nodo n_1 y las variables asociadas a él, cuando $s_{z_3}.setPredToInf_{n_1} = \emptyset$. La ejecución de la acción $rcvINF_{n_1}(n)$ en s_{z_3} $(z > z_3 > z_2)$ trae consigo cambios en las variables del nodo n_1: $s_{z_3}.status.id_{n_1} = known$ y $s_{z_3}.st\_alg_{n_1} = NULL$. Como en s_{z_3} persiste el valor *false* de $s_{z_3}.nofirstAVS_{n_1}$, habrá que esperar a que el nodo n_1 pase a ser *candidate* mediante la acción $initiate_{n_1}$ para que finalmente $s_z.nofirstAVS_{n_1} = true$ y $s_z.status.alg_{n_1} = candidate$ (condiciones indispensables para aplicar la propiedad B.2.72). Al alcanzar el nodo n_1 el estado *candidate*, ya estará en disposición de recibir el mensaje ALG que debe enviar el nodo n_j.

Opción 4:

Inicialmente el nodo n_1 formaba parte de un camino en el que está incluida la ruta p. En s_{z_1} se ejecutó la acción $firstAVS_{n_1}$, siendo $z_1 < z$ y $s_{z_1}.sim\_id_{n_1} > s_{z_1}.sim\_id_{n_l}$. Los efectos de esta acción consistieron en la creación de un mensaje AVS dirigido a un candidato n_l y en la modificación de la variable $nofirstAVS_{n_1}$, de manera que $s_{z_1}.nofirstAVS_{n_1} = false$. Como la ruta almacenada en $s_{z_1}.st\_alg_{n_1}$ coincide parcialmente con la ruta p de s_z (p está incluida en $s_{z_1}.st\_alg_{n_1}.path$), se puede afirmar que el nodo n_j está incluido en ambas. Eso significa que el nodo n_j pasó a ser *dummy* cuando reenvió el mensaje ALG que dio lugar al mensaje AVS que mandó el nodo n_1 en s_{z_1}. Para conseguir que n_j pase a ser *candidate* en el camino descrito por p es imprescindible que el nodo n_l ejecute la acción $EndDelArc_{n_l}(n\prime\prime)$. De esta forma, las esperas entre n_j y n_l podrán ser liberadas, consiguiendo que en la ruta p haya sólo dos candidatos ($L = 2$).

Se sabe que el mensaje AVS, por la propiedad de equidad, llega al nodo n_l. En este punto se analiza un abanico de posibles ejecuciones en función de las acciones

necesarias para llegar a $L = 2$. Muchas acciones se ejecutan porque ninguna otra puede deshabilitarla y la partición obliga a ello:

Caso 1: se ejecuta $rcvAVS_{n_l}(n_1, m)$ antes de activarse el nodo n_l.
Secuencia de acciones 1: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *known*.

- $\pi_{z_2} = rcvAVS_{n_l}(n_1, m)$, $z_1 < z_2 < z$. El mensaje *AVS* se almacena en $s_{z_2}.set\_st\_avs_{n_l}$, siendo $m.sid = s_{z_1}.sim\_id_{n_1}$.

- $\pi_{z_3} = EndDelArc_{n_l}(n\prime\prime)$, $z_2 < z_3 < z$. El nodo n_l se activa y cambia su tiempo de activación. Esta ejecución supone que $s_{z_3}.status.id_{n_l} = known$.

- $\pi_{z_4} = StartDelArc_{n_l}(n\prime, t_{n\prime})$, $z_3 < z_4 < z$. Si $n\prime$ es el único predecesor de n_l, los efectos de la acción consisten en un mensaje *INF* con $m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$ y un mensaje *AVS*, n, tal que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ (la ruta del mensaje ya no contiene al nodo n_l). Además, $s_{z_4}.sim\_id_{n_l} = (n_l, s_{z_4}.ta_{n_l}, \epsilon)$, siendo evidente que $s_{z_3}.sim\_id_{n_l} < s_{z_4}.sim\_id_{n_l}$ porque el tiempo de activación del nodo n_l ha aumentado.

  Cuando n_l tiene más predecesores, se reenvía el contenido de $s_{z_3}.set\_st\_avs_{n_l}$ en forma de mensaje *AVS*. El mensaje cumple que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ y la ruta del mensaje deja de contener al nodo n_l. A su vez, el nodo n_l manda un mensaje *INF* a $n\prime$, siendo $m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$. Por otro lado, $s_{z_4}.sim\_id_{n_l} = s_{z_3}.sim\_id_{n_l}$.

  Los mensajes *AVS* que se retiran de $s_{z_3}.set\_st\_avs_{n_l}$ deben cumplir siempre que el penúltimo elemento de la ruta sea precisamente el nodo $n\prime$.

- En este punto de la ejecución cabe destacar que el mensaje *INF* no podrá ser procesado hasta que se ejecute $EndDelArc_{n\prime}(n_l)$ y que la recepción del mensaje *AVS* con destino $n\prime$ requiere que se haya ejecutado previamente $EndDelArc_{n\prime}(n_l)$ y $rcvINF_{n\prime}(n_l, m)$. La acción $StartDelArc_{n\prime}(n, t_n)$ también estará habilitada una vez que se hayan producido los efectos de la acción $EndDelArc_{n\prime}(n_l)$.

- $\pi_{z_5} = EndDelArc_{n\prime}(n_l)$, $z_4 < z_5 < z$. Los efectos de esta acción modifican las siguientes variables: $s_{z_5}.status.alg_{n\prime} = active$ y $s_{z_5}.status.id_{n\prime} = unknown$. El tiempo de activación del nodo $n\prime$ también cambia al aumentar en una unidad.

- $\pi_{z_6} = rcvINF_{n\prime}(n_l, m)$, $z_5 < z_6 < z$. Al ejecutarse esta acción: $s_{z_6}.status.id_{n\prime} = known$ y $s_{z_6}.sim\_id_{n\prime} = m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$. Si, tras la ejcución de la acción $\pi_{z_5} = EndDelArc_{n\prime}(n_l)$, se hubiera ejecutado $StartDelArc_{n\prime}(n, t_n)$, surgiría un nuevo mensaje *INF* con destino al nodo n con $m\prime_{INF}.sid = s_{z_3}.sim\_id_{n_l}$.

- $\pi_{z_7} = rcvAVS_{n\prime}(n_l,\ m)$, $z_6 < z_7 < z$. La acción está habilitada debido a que el nodo $n\prime$ se ha desbloqueado, adquiriendo un nuevo tiempo de activación e impidiendo así que el tiempo del nodo $n\prime$ coincida con el que tiene asociado en la ruta del mensaje *AVS*. Al ejecutar esta acción, es posible que el mensaje *AVS* se almacene en $s_{z_7}.set\_st\_avs_{n\prime}$ ($penult(m.path) \in s_{z_7-1}.setPred_{n\prime}$) o se reenvíe con $n.fwd = true$ y $n.sid = s_{z_1}.sim\_id_{n_1}$ (si ya ha sido ejecutada la acción $StartDelArc_{n\prime}(n,\ t_n)$ y, por lo tanto, $penult(m.path) \notin s_{z_7-1}.setPred_{n\prime}$).

- La ejecución de acciones $EndDelArc_x(y)$, $rcvINF_x(y,\ m)$ y $rcvAVS_x(y,\ m)$ se sucederá hasta alcanzar el nodo n_j. Hay que señalar que la acción $StartDelArc_x(y,\ t_y)$ se podrá ejecutar después de cualquiera de ellas, dando lugar a distintas formas de propagar el mensaje *AVS*.

- $\pi_{z_8} = EndDelArc_{n_j}(n_k)$, $z_7 < z_8 < z$. Los efectos de esta acción consisten en: $s_{z_8}.status.id_{n_j} = unknown$, $s_{z_8}.ta_{n_j} > s_{z_7}.ta_{n_j}$ y $s_{z_8}.status.alg_{n_j} = active$. El nodo n_j quedará a la espera de recibir un mensaje *INF* que contenga la nueva identidad simulada que tendrá que adoptar obligatoriamente por ser *dummy* anteriormente.

- $\pi_{z_9} = StartAddArc_{n_j}(n)$, $z_8 < z_9 < z$. El nodo n_j cambiará su estado $s_{z_9}.status.alg_{n_j} = blocked$, pero se mantendrá como $s_{z_9}.status.id_{n_j} = unknown$. Para alcanzar los requisitos del antecedente de la propiedad, será necesario que n_j se convierta en *candidate*.

- $\pi_{z_{10}} = rcvINF_{n_j}(n_k)$, $z_9 < z_{10} < z$. Cuando el nodo n_j recibe el mensaje *INF*, éste cambia siempre su identidad simulada porque antes de activarse se comportaba como un nodo *dummy*. La nueva identidad simulada es, por tanto, $s_{z_{10}}.sim\_id_{n_j} = m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$. Al mismo tiempo, se tiene que $s_{z_{10}}.status.id_{n_j} = known$ y $s_{z_{10}}.status.alg_{n_j} = candidate$ porque $s_{z_{10}}.setPredToInf_{n_j} \neq \emptyset$.

- $\pi_{z_{11}} = rcvAVS_{n_j}(n_k,\ m)$, $z_{10} < z_{11} < z$. El mensaje *AVS*, finalmente alcanza el nodo n_j. Hay que recordar que se está considerando el caso en el que $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$. Esto implica que no se puede verificar $s_{z_{11}-1}.sim\_id_{n_j} < m_{AVS}.sid$ porque $m_{AVS}.sid = s_{z_1}.sim\_id_{n_1}$ y $s_{z_{11}-1}.sim\_id_{n_j} = s_{z_3}.sim\_id_{n_l}$. La condición que permitió la ejecución de la acción $firstAVS_{n_1}$ consistía en: $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_l}$. Así que, se llega a una contradicción en la que se evidencia que es imposible que en la configuración de nodos impuesta se produzca el esquema de ejecución analizado.

Secuencia de acciones 2: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *unknown* recibiendo una identidad simulada que es mayor que la suya, pero $m_{INF}.sid > n.sid = s_{z_2}.sim\_id_{n_1}$.

- $\pi_{z_2} = rcvAVS_{n_l}(n_1,\ m)$, $z_1 < z_2 < z$. El mensaje *AVS* pasa a almacenarse en $s_{z_2}.set\_st\_avs_{n_l}$, siendo $m.sid = s_{z_1}.sim\_id_{n_1}$.

- $\pi_{z_3} = EndDelArc_{n_l}(n'')$, $z_2 < z_3 < z$. El nodo n_l se activa y cambia su tiempo de activación. Esta ejecución supone que $s_{z_3}.status.id_{n_l} = unknown$ y queda pendiente de recibir un mensaje *INF*.

- $\pi_{z_4} = rcvINF_{n_l}(x,\ m)$, $z_3 < z_4 < z$. Si el nodo n_l, anteriormente *candidate*, recibe un mensaje *INF*, siendo $m_{INF}.sid > s_{z_4-1}.sim\_id_{n_l}$, éste cambia su identidad simulada de manera que $s_{z_4}.sim\_id_{n_l} = m_{INF}.sid$.

  Considerando que se produce este cambio de identidad simulada y que en $s_{z_4-1}.set\_st\_avs_{n_l}$ hay un mensaje tal que $m_{INF}.sid > n.sid = s_{z_2}.sim\_id_{n_1}$, los efectos de la acción eliminarán el mensaje de ese almacén y se formará un mensaje *AVSRSP* a partir de él. El mensaje *AVSRSP* tiene la misma ruta que el mensaje de $s_{z_4-1}.set\_st\_avs_{n_l}$ y va dirigido al primer nodo de la ruta, n_1, con tiempo correcto. La identidad simulada es $s_{z_4}.sim\_id_{n_l} = m_{INF}.sid$ y se corresponde con la de un nodo ajeno a la ruta que estaba almacenada en $s_{z_1}.st\_alg_{n_1}.path$.

- La acción $\pi_{z_5} = rcvAVSRSP_{n_1}(n_l,\ m)$, $z_4 < z_5 < z$ está habilitada en cualquier instante porque el nodo n_1 no se desbloquea y mantiene constante su tiempo de activación. Los efectos de esta acción permitirán que se almacene una ruta en $s_{z_5}.st\_avsrsp_{n_1}$ que se caracteriza por los siguientes campos: $m.ta = s_z.ta_{n_1}$, $m.sid = (x,\ t,\ nu)$ y $m.path = n_{AVS}.path = (n_1,\ s_z.ta_{n_1})\ \ldots\ (n_j,\ t_j)\ \ldots\ (n_l,\ t_l)$, donde $x = s_{z_4}.sim\_id_{n_l}.id$, $t = s_{z_4}.sim\_id_{n_l}.ta$ y $nu = s_{z_4}.sim\_id_{n_l}.nu$. A partir de este mensaje *AVSRSP* el nodo n_1 llegará a conocer al mayor candidato sucesor, n_j.

  El nodo n_l también tendría oportunidad de enviar un mensaje *INF* al nodo n' si ya hubiera ejecutado la acción $StartDelArc_{n_l}(n',\ t)$ pero éste no es el caso.

  Como el mensaje *AVS* almacenado en $s_{z_2}.set\_st\_avs_{n_l}$ ha sido tratado y el nodo n_1 va a almacenarlo en $s_{z_5}.st\_avsrsp_{n_1}$, las identidades simuladas de los nodos entre n_j y n_l van cambiando y adquiriendo la identidad simulada que tomó el nodo n_l. Este proceso se realizará mediante la ejecución de las acciones $EndDelArc_x(y)$, $rcvINF_x(y,\ m)$ y $StartDelArc_x(y,\ t)$, pudiéndose ésta última adelantarse a la acción $rcvINF_x(y,\ m)$.

- $\pi_{z_6} = EndDelArc_{n_j}(n_k)$, $z_5 < z_6 < z$. Los efectos de esta acción consisten en: $s_{z_6}.status.id_{n_j} = unknown$, $s_{z_6}.ta_{n_j} > s_{z_5}.ta_{n_j}$ y $s_{z_6}.status.alg_{n_j} = active$. El nodo n_j quedará a la espera de recibir un mensaje *INF* que contenga la nueva identidad simulada, que obligatoriamente tendrá que adoptar por haber sido anteriormente un nodo *dummy*.

- $\pi_{z_7} = StartAddArc_{n_j}(n)$, $z_6 < z_7 < z$. El nodo n_j cambiará su estado $s_{z_7}.status.alg_{n_j} = blocked$, pero se mantendrá como $s_{z_7}.status.id_{n_j} = unknown$. Para alcanzar los requisitos del antecedente de la propiedad, será necesario que n_j se convierta en *candidate*.

- $\pi_{z_8} = rcvINF_{n_j}(n_k)$, $z_7 < z_8 < z$. Cuando el nodo n_j recibe el mensaje *INF*, éste cambia siempre su identidad simulada porque antes de activarse se comportaba como un nodo *dummy*. La nueva identidad simulada es, por tanto, $s_{z_8}.sim\_id_{n_j} = m_{INF}.sid = s_{z_4}.sim\_id_{n_l}$. Al mismo tiempo, se tiene que $s_{z_8}.status.id_{n_j} = known$ y $s_{z_8}.status.alg_{n_j} = candidate$ porque $s_{z_8}.setPredToInf_{n_j} \neq \emptyset$.

- El orden de estas dos últimas acciones se puede invertir: $\pi_{z_7} = rcvINF_{n_j}(n_k)$, $z_6 < z_7 < z$ y $\pi_{z_8} = StartAddArc_{n_j}(n)$, $z_7 < z_8 < z$. En esta otra secuencia de ejecución, el nodo n_j pasa a ser *candidate* por efecto de la acción $\pi_{z_8} = StartAddArc_{n_j}(n)$. En conclusión, la ejecución planteada es posible.

Secuencia de acciones 3: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el n_l pasa a ser *unknown* recibiendo una identidad simulada que es mayor que la suya, pero $m_{INF}.sid \leq n.sid = s_{z_2}.sim\_id_{n_1}$.

- Las acciones $\pi_{z_2} = rcvAVS_{n_l}(n_1, m)$, $z_1 < z_2 < z$, $\pi_{z_3} = EndDelArc_{n_l}(n\prime\prime)$, $z_2 < z_3 < z$ y $\pi_{z_4} = rcvINF_{n_l}(x, m)$, $z_3 < z_4 < z$ se ejecutan al igual que el apartado anterior. Los efectos de la última acción cambian la identidad simulada del nodo n_l ($s_{z_4}.sim\_id_{n_l} = m_{INF}.sid$), pero el mensaje almacenado se eliminará de $set\_st\_avs_{n_l}$ al ejecutar la acción $StartDelArc_{n_l}(n\prime, t)$.

- $\pi_{z_5} = StartDelArc_{n_l}(n\prime, t)$, $z_4 < z_5 < z$. El efecto de interés de esta acción, posea n_l uno o más predecesores, es reenviar el contenido de $s_{z_3}.set\_st\_avs_{n_l}$ como un mensaje *AVS*, n. El mensaje cumple que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ y su ruta deja de contener al nodo n_l. Los mensajes *AVS* que se retiran de $s_{z_3}.set\_st\_avs_{n_l}$ deben cumplir que el penúltimo elemento de la ruta sea precisamente el nodo $n\prime$. También surge un mensaje *INF* con identidad simulada $s_{z_4}.sim\_id_{n_l}$ y destino el nodo $n\prime$. Ese mensaje se recibirá al activarse $n\prime$.

- $\pi_{z_6} = EndDelArc_{n\prime}(n_l)$, $z_5 < z_6 < z$. El nodo $n\prime$ se activa pero $s_{z_6}.status.id = unknown$. Esto impedirá que el nodo $n\prime$ procese los mensajes *AVS* que le van dirigidos.

- $\pi_{z_7} = rcvINF_{n\prime}(n_l, m)$, $z_6 < z_7 < z$. El nodo $n\prime$ cambia su identidad simulada y la difunde entre los nodos que pertenecen sólo a su *setPredToInf*. Además, $s_{z_7}.status.id_{n\prime} = known$.

- $\pi_{z_8} = rcvAVS_{n\prime}(n_l,\ m)$, $z_7 < z_8 < z$. Una vez habilitada la recepción del mensaje *AVS* ($s_{z_7}.status.id_{n\prime} = known$), éste podrá ser reenviado si el nodo $n\prime$ ya se ha activado y ha eliminado al siguiente predecesor de la ruta.

- $\pi_{z_9} = StartDelArc_{n_k}(n_j,\ t)$, $z_8 < z_9 < z$. Al retirar el nodo n_j de $setPred_{n_k}$, se eliminan los mensajes *AVS* que pueda contener $s_{z_9-1}.set\_st\_avs_{n_k}$, siempre y cuando su penúltimo elemento sea precisamente el nodo n_j.

- $\pi_{z_{10}} = EndDelArc_{n_j}(n_k)$, $z_9 < z_{10} < z$. Al ejecutarse esta acción: $s_{z_{11}}.status.alg_{n_j} = active$ y $s_{z_{11}}.status.id_{n_j} = unknown$. El tiempo de activación también se incrementa.

- $\pi_{z_{11}} = StartAddArc_{n_j}(n)$, $z_{10} < z_{11} < z$. Aunque el nodo n_j cambia de estado, $s_{z_{11}}.status.alg_{n_j} = blocked$, $s_{z_{11}}.status.id_{n_j}$ sigue siendo *unknown*.

- $\pi_{z_{12}} = rcvINF_{n_j}(n_k)$, $z_{11} < z_{12} < z$. Los efectos de esta acción consisten en un cambio de identidad simulada, $s_{z_{13}}.sim\_id_{n_j} = m_{INF}.sid = s_{z_4}.sim\_id_{n_l}$. Además, $s_{z_{13}}.status.id_{n_j} = known$ y $s_{z_{13}}.status.alg_{n_j} = candidate$.

- $\pi_{z_{13}} = rcvAVS_{n_j}(n_k,\ m)$, $z_{12} < z_{13} < z$. El mensaje *AVS* generado en z_9 ya podrá ser tratado porque el nodo n_j es *known*. Sabiendo que: $s_z.sim\_id_{n_1} < s_z.sim\_id_{n_j}$, $m.sid = s_z.sim\_id_{n_1}$ y $s_z.sim\_id_{n_j} = s_{z_4}.sim\_id_{n_l}$, se llega a la conclusión de que $m.sid < s_{z_4}.sim\_id_{n_l}$. Esta relación evidencia que existe una contradicción de acuerdo con la restricción impuesta en este apartado, $m_{INF}.sid = s_{z_4}.sim\_id_{n_l} \leq m.sid = s_z.sim\_id_{n_1}$. Esto implica que la ejecución planteada es imposible.

Secuencia de acciones 4: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *unknown* y la identidad simulada que recibe no es mayor que la suya, es decir, $m_{INF}.sid \leq s_{z_4-1}.sim\_id_{n_l}$:

- Las acciones $\pi_{z_2} = rcvAVS_{n_l}(n_1,\ m)$, $z_1 < z_2 < z$, $\pi_{z_3} = EndDelArc_{n_l}(n\prime\prime)$, $z_2 < z_3 < z$ se ejecutan al igual que en el apartado anterior.

- $\pi_{z_4} = rcvINF_{n_l}(x,\ m)$, $z_3 < z_4 < z$. Si el nodo n_l, anteriormente *candidate*, recibe un mensaje *INF*, siendo $m_{INF}.sid \leq s_{z_4-1}.sim\_id_{n_l}$, éste mantiene su identidad simulada de manera que $s_{z_4}.sim\_id_{n_l} = s_{z_4-1}.sim\_id_{n_l}$.

  Al ejecutar esta acción, se producen otros efectos: $s_{z_4}.status.id_{n_l} = known$ y $s_{z_4}.sim\_id_{n_l} = s_{z_4-1}.sim\_id_{n_l}$ (se ignora la identidad simulada que trae el mensaje *INF*). Si, tras la ejcución de la acción $\pi_{z_3} = EndDelArc_{n_l}(n\prime\prime)$, se hubiera ejecutado $StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, surgiría un nuevo mensaje *INF* hacia el nodo $n\prime$ con $m\prime_{INF}.sid = s_{z_4}.sim\_id_{n_l}$.

- $\pi_{z_5} = StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, $z_4 < z_5 < z$. Si $n\prime$ es el único predecesor de n_l, los efectos de esta acción generan un mensaje *INF* con $m_{INF}.sid = s_{z_4}.sim\_id_{n_l}$ y un mensaje *AVS*, n, tal que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ (la ruta del mensaje ya no contiene al nodo n_l). Además, $s_{z_5}.sim\_id_{n_l} = (n_l,\ s_{z_5}.ta_{n_l},\ \epsilon)$ porque $s_{z_3}.sim\_id_{n_l} < s_{z_4}.sim\_id_{n_l}$ (el tiempo de activación del nodo n_l se ha incrementado).

  Cuando n_l tiene más predecesores, se reenvía el contenido de $s_{z_3}.set\_st\_avs_{n_l}$ en forma de mensaje *AVS*. El mensaje cumple que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ y su ruta omite el nodo n_l. Además, el nodo n_l manda un mensaje *INF* a $n\prime$, siendo $m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$. Por otro lado, $s_{z_4}.sim\_id_{n_l} = s_{z_3}.sim\_id_{n_l}$.

  Por otra parte, los mensajes *AVS* que se retiran de $s_{z_3}.set\_st\_avs_{n_l}$ siempre deben cumplir que el penúltimo elemento de su ruta sea precisamente el nodo $n\prime$.

- Una vez ejecutada la acción $\pi_{z_5} = StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, la secuencia posible de acciones es similar a la descrita en la *Secuencia de acciones 1* de este mismo Caso 1. La conclusión que se obtiene en esa situación es igualmente válida para la configuración que aquí se plantea.

Caso 2: se activa el nodo n_l antes de ejecutarse la acción $rcvAVS_{n_l}(n_1,\ m)$.

Secuencia de acciones 1: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *known*.

- $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, $z_1 < z_2 < z$. El nodo n_l se activa y cambia su tiempo de activación. Esta ejecución supone que $s_{z_2}.status.id_{n_l} = known$.

- $\pi_{z_3} = rcvAVS_{n_l}(n_1,\ m)$, $z_2 < z_3 < z$. Como $s_{z_3-1}.status.id_{n_l} = known$, $m.fwd = false$, $s_{z_3-1}.sim\_id_{n_l} < s_{z_3-1}.sim\_id_{n_1}$ y $penult(m.path) \in s_{z_3-1}.setPred_{n_l}$, el mensaje *AVS* se almacena en $s_{z_3}.set\_st\_avs_{n_l}$, siendo $m.sid = s_{z_1}.sim\_id_{n_1}$.

- Después de la ejecución de la acción $\pi_{z_3} = rcvAVS_{n_l}(n_1,\ m)$, se reproduce la *Secuencia de acciones 1* correspondiente al Caso 1, concluyendo que la situación estudiada también es imposible.

Secuencia de acciones 2: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *known*. Además, antes de recibir el mensaje *AVS*, se ejecuta la acción $StartDelArc_{n_l}(n\prime,\ t_{n\prime})$.

- $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, $z_1 < z_2 < z$. El nodo n_l se activa y cambia su tiempo de activación. En esta ejecución, se va a suponer que $s_{z_2}.status.id_{n_l} = known$.

- $\pi_{z_3} = StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, $z_2 < z_3 < z$. Si $n\prime$ es el único predecesor de n_l, los efectos de esta acción generan un mensaje *INF* con $m_{INF}.sid = s_{z_3-1}.sim\_id_{n_l}$ y cambian la identidad simulada del nodo n_l de forma que $s_{z_3}.sim\_id_{n_l} = (n_l,\ s_{z_3}.ta_{n_l},\ \epsilon)$. Es evidente que $s_{z_2}.sim\_id_{n_l} < s_{z_3}.sim\_id_{n_l}$ porque el tiempo de activación del nodo n_l se incrementó al activarse.

  Cuando n_l tiene más predecesores, el nodo n_l también manda un mensaje *INF* a $n\prime$, siendo $m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$, pero $s_{z_2}.sim\_id_{n_l} = s_{z_3}.sim\_id_{n_l}$.

- $\pi_{z_4} = rcvAVS_{n_l}(n_1,\ m)$, $z_3 < z_4 < z$. El mensaje *AVS* ya no se almacena en $s_{z_2}.set\_st\_avs_{n_l}$ porque, aunque $m.fwd = false$, resulta que $penult(m.path) \notin s_{z_4-1}.setPred_{n_l}$. En consecuencia, el efecto de esta acción sólo consiste en reenviar el mensaje *AVS* al nodo $n\prime$ con el campo *fwd* a *true*, en el campo *sid* el valor $s_{z_1}.sim\_id_{n_1}$ y una ruta en la que ya no aparece el nodo n_l.

- En este punto de la ejecución cabe destacar que el mensaje *INF* no podrá ser procesado hasta que se ejecute $EndDelArc_{n\prime}(n_l)$ y que el mensaje *AVS* con destino $n\prime$ requiere que se haya ejecutado previamente $EndDelArc_{n\prime}(n_l)$ y $rcvINF_{n\prime}(n_l,\ m)$ para su recepción. La acción $StartDelArc_{n\prime}(n,\ t_n)$ también estará habilitada una vez que se hayan producido los efectos de la acción $EndDelArc_{n\prime}(n_l)$. A partir de este momento de la ejecución, se procede como en la *Secuencia de acciones 1* correspondiente al Caso 1 y se concluye del mismo modo.

Secuencia de acciones 3: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *unknown* y la identidad simulada que recibe el nodo n_l es mayor que la suya $m_{INF}.sid > s_{z_1}.sim\_id_{n_l}$, pero $m_{INF}.sid > s_{z_1}.sim\_id_{n_1}$.

- La acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, $z_1 < z_2 < z$, se ejecuta al igual que en otros apartados de este mismo caso.

- $\pi_{z_3} = rcvINF_{n_l}(n\prime\prime,\ m)$, $z_2 < z_3 < z$. Si el nodo n_l, anteriormente *candidate*, recibe un mensaje *INF*, siendo $m_{INF}.sid > s_{z_3-1}.sim\_id_{n_l}$, el nodo n_l adopta esa nueva identidad simulada.

  Al ejecutar esta acción, se producen los siguientes efectos: $s_{z_3}.status.id_{n_l} = known$ y $s_{z_3}.sim\_id_{n_l} = m_{INF}.sid$. Si, tras la ejcución de la acción π_{z_2}, se hubiera ejecutado $StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, surgiría un nuevo mensaje *INF* con destino el nodo $n\prime$ y $m\prime_{INF}.sid = s_{z_3}.sim\_id_{n_l}$.

- $\pi_{z_4} = rcvAVS_{n_l}(n_1,\ m)$, $z_3 < z_4 < z$. El mensaje *AVS* se transforma en un mensaje de tipo *AVSRSP* porque $s_{z_4-1}.sim\_id_{n_l} > s_{z_4-1}.sim\_id_{n_1}$. El mensaje *AVSRSP* tiene la misma ruta que el mensaje *AVS* recibido, va dirigido al primer nodo de la ruta, n_1, y tiene tiempo correcto e identidad simulada $s_{z_3}.sim\_id_{n_l}$

$= m_{INF}.sid$. Esa identidad corresponde a un nodo ajeno a la ruta que estaba almacenada en $s_{z_1}.st\_alg_{n_1}.path$.

- Si, tras la ejcución de la acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, se hubiera ejecutado $StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, $penult(m.path) \notin s_{z_3-1}.setPred_{n_l}$, el efecto de la acción $rcvAVS_{n_l}(n_1,\ m)$, habría consistido en reenviar el mensaje *AVS* al nodo $n\prime$ con el campo *fwd* a *true*, el campo *sid* cargado con el valor $s_{z_1}.sim\_id_{n_1}$ y una ruta modificada sin el nodo n_l. Esto implica que la conversión del mensaje *AVS* en *AVSRSP* se pospondría al momento en el que la acción $rcvAVS_x(y,\ m)$ se adelantara a la acción $StartDelArc_x(y,\ t)$.

- La acción $\pi_{z_5} = rcvAVSRSP_{n_1}(n_l,\ m)$, $z_4 < z_5 < z$ está habilitada en cualquier instante porque el nodo n_1 no se desbloquea y mantiene constante su tiempo de activación. Los efectos de esta acción permitirán que se almacene una ruta en $s_{z_5}.st\_avsrsp_{n_1}$ que se caracteriza por contener los siguientes campos: $m.ta = s_z.ta_{n_1}$, $m.sid = (x,\ t,\ nu)$ y $m.path = n_{AVS}.path = (n_1,\ s_z.ta_{n_1}) \ldots (n_j,\ t_j) \ldots (n_l,\ t_l)$, siendo $x = s_{z_4}.sim\_id_{n_l}.id$, $t = s_{z_4}.sim\_id_{n_l}.ta$ y $nu = s_{z_4}.sim\_id_{n_l}.nu$.

  A partir de este mensaje *AVSRSP* el nodo n_1 llegará a conocer al candidato mayor que le sucede, n_j. Como el nodo n_1 va a almacenar el mensaje *AVSRSP* en $s_{z_5}.st\_avsrsp_{n_1}$, sólo resta que las identidades simuladas de los nodos entre n_j y n_l vayan cambiando y adquiriendo la identidad simulada que tomó el nodo n_l. Este proceso se realizará mediante la ejecución de las acciones $EndDelArc_x(y)$, $rcvINF_x(y,\ m)$ y $StartDelArc_x(y,\ t)$, pudiéndose ésta última adelantarse a la acción $rcvINF_x(y,\ m)$.

- La ejecución evoluciona, a partir de este punto, como en la *Secuencia de acciones 2* del Caso 1: $\pi_{z_6} = EndDelArc_{n_j}(n_k)$ $(z_5 < z_6 < z)$, $\pi_{z_7} = StartAddArc_{n_j}(n)$ $(z_6 < z_7 < z)$ y $\pi_{z_8} = rcvINF_{n_j}(n_k)$ $(z_7 < z_8 < z)$.

Secuencia de acciones 4: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *unknown* y la identidad simulada que recibe este nodo es mayor que la suya, es decir, $m_{INF}.sid > s_{z_3-1}.sim\_id_{n_l}$, pero $m_{INF}.sid < s_{z_1}.sim\_id_{n_1}$.

- La acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, $z_1 < z_2 < z$, se ejecuta al igual que en otros apartados de este mismo caso.

- $\pi_{z_3} = rcvINF_{n_l}(n\prime\prime,\ m)$, $z_2 < z_3 < z$. Si el nodo n_l, anteriormente *candidate*, recibe un mensaje *INF*, siendo $m_{INF}.sid > s_{z_3-1}.sim\_id_{n_l}$, el nodo n_l adopta esa nueva identidad simulada.

  Al ejecutar esta acción, se producen los siguientes efectos: $s_{z_3}.status.id_{n_l} = known$ y $s_{z_3}.sim\_id_{n_l} = m_{INF}.sid$. Si, tras la ejcución de la acción π_{z_2}, se hubiera ejecutado $StartDelArc_{n_l}(n\prime,\ t_{n\prime})$, habría surgido un nuevo mensaje *INF* con destino al nodo $n\prime$ y $m\prime_{INF}.sid = s_{z_3}.sim\_id_{n_l} = m_{INF}.sid$.

- $\pi_{z_4} = rcvAVS_{n_l}(n_1, m)$, $z_3 < z_4 < z$. Dado que $s_{z_4-1}.status.id_{n_l} = known$, $m.fwd = false$, $s_{z_4-1}.sim\_id_{n_l} < s_{z_1}.sim\_id_{n_1}$ y $penult(m.path) \in s_{z_4-1}.setPred_{n_l}$, siendo $s_{z_3-1}.sim\_id_{n_l} < s_{z_4-1}.sim\_id_{n_l}$, el mensaje *AVS* se almacena en $s_{z_4}.set\_st\_avs_{n_l}$, siendo $m.sid = s_{z_1}.sim\_id_{n_1}$

- Si, tras la ejcución de la acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, se hubiera ejecutado $StartDelArc_{n_l}(n\prime, t_{n\prime})$, $penult(m.path) \notin s_{z_3-1}.setPred_{n_l}$ y, por lo tanto, el mensaje *AVS* ya no se almacenaría en $s_{z_4}.set\_st\_avs_{n_l}$ aunque fuera $m.fwd = false$. En consecuencia, el efecto de la acción $rcvAVS_{n_l}(n_1, m)$, en esta situación, sólo consistiría en reenviar el mensaje *AVS* al nodo $n\prime$ con el campo *fwd* a *true*, el campo *sid* cargado con el valor $s_{z_1}.sim\_id_{n_1}$ y con una ruta modificada en la que ya no aparecería el nodo n_l.

- $\pi_{z_5} = StartDelArc_{n_l}(n\prime, t_{n\prime})$, $z_4 < z_5 < z$. Si $n\prime$ es el único predecesor de n_l, los efectos de esta acción crean en un mensaje *INF* al nodo $n\prime$ con $m_{INF}.sid = s_{z_5-1}.sim\_id_{n_l}$ ($s_{z_5-1}.sim\_id_{n_l} = s_{z_4}.sim\_id_{n_l} > s_{z_3-1}.sim\_id_{n_l}$). Además, se genera un mensaje *AVS*, n, tal que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ (la ruta del mensaje ya no contiene al nodo n_l)y $s_{z_5}.sim\_id_{n_l} = (n_l, s_{z_5}.ta_{n_l}, \epsilon)$. Es evidente que $s_{z_4}.sim\_id_{n_l} < s_{z_5}.sim\_id_{n_l}$ porque el tiempo de activación del nodo n_l ha aumentado.

  Cuando n_l tiene más predecesores, se reenvía el contenido de $s_{z_4-1}.set\_st\_avs_{n_l}$ en forma de mensaje *AVS*. El mensaje cumple que $n.fwd = true$, $n.sid = s_{z_1}.sim\_id_{n_1}$ y la ruta del mensaje deja de contener al nodo n_l. Además, el nodo n_l manda un mensaje *INF* al nodo $n\prime$, siendo $m_{INF}.sid = s_{z_3}.sim\_id_{n_l}$. Por otro lado, $s_{z_5}.sim\_id_{n_l} = s_{z_4}.sim\_id_{n_l} > s_{z_3-1}.sim\_id_{n_l}$.

  Los mensajes *AVS* que se retiran de $s_{z_5-1}.set\_st\_avs_{n_l}$ deben cumplir siempre que el penúltimo elemento de la ruta sea precisamente el nodo $n\prime$.

- Una vez ejecutada la acción $\pi_{z_5} = StartDelArc_{n_l}(n\prime, t_{n\prime})$, la secuencia de acciones es similar a la que se ha descrito para la *Secuencia de acciones 1* del Caso 1. Por consiguiente, la conclusión alcanzada en ese caso también es válida para la ejecución planteada en este apartado.

Secuencia de acciones 5: Tras ejecutar $EndDelArc_{n_l}(n\prime\prime)$, el nodo n_l pasa a ser *unknown* y la identidad simulada que recibe no es superior a la suya, es decir, $m_{INF}.sid \leq s_{z_4-1}.sim\_id_{n_l}$. Obviamente $m_{INF}.sid < s_{z_1}.sim\_id_{n_1}$:

- La acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, $z_1 < z_2 < z$, se ejecuta al igual que en otros apartados de este mismo caso.

- $\pi_{z_3} = rcvINF_{n_l}(n\prime\prime, m)$, $z_2 < z_3 < z$. Si el nodo n_l, anteriormente *candidate*, recibe un mensaje *INF*, siendo $m_{INF}.sid \leq s_{z_3-1}.sim\_id_{n_l}$, éste mantiene su identidad simulada de manera que $s_{z_3}.sim\_id_{n_l} = s_{z_3-1}.sim\_id_{n_l}$.

  Al ejecutar esta acción, se producen los siguientes efectos: $s_{z_3}.status.id_{n_l} = known$ y $s_{z_3}.sim\_id_{n_l} = s_{z_3-1}.sim\_id_{n_l}$ (se ignora la identidad simulada que trae el mensaje *INF*). Si, tras la ejcución de la acción $\pi_{z_2} = EndDelArc_{n_l}(n\prime\prime)$, se hubiera ejecutado $StartDelArc_{n_l}(n\prime, t_{n\prime})$, se habrí creado un nuevo mensaje *INF* con destino el nodo $n\prime$ y con $m\prime_{INF}.sid = s_{z_3-1}.sim\_id_{n_l}$.

- $\pi_{z_4} = rcvAVS_{n_l}(n_1, m)$, $z_3 < z_4 < z$. El mensaje *AVS* se almacena en $s_{z_4}.set\_st\_avs_{n_l}$, siendo $m.sid = s_{z_1}.sim\_id_{n_1}$, porque $s_{z_4-1}.status.id_{n_l} = known$. Además, $m.fwd = false$, $s_{z_4-1}.sim\_id_{n_l} < s_{z_1}.sim\_id_{n_1}$ y $penult(m.path) \in s_{z_4-1}.setPred_{n_l}$.

- A partir de la ejecución de $\pi_{z_4} = rcvAVS_{n_l}(n_1, m)$, la secuencia de acciones posible coincide con la *Secuencia de acciones 4* de este mismo Caso 2, siendo válida la misma conclusión que se extrae en esa situación.

En cualquiera de los esquemas de ejecución posibles $s_z.status.alg_{n_j} = candidate$ y $s_z.nofirstAVS_{n_1} = \{true, false\}$, existe la ruta p que cumple las condiciones de un camino $wait(n_1, n_j, p, s_z)$ y $s_z.st\_alg_{n_1}.path = \gamma_1 \cdot (n_j, t_j) \cdot \gamma_2$ y $p = (n_1, t_1) \cdot \gamma_1 \cdot (n_j, t_j)$. Además, en $s_z.st\_avsrsp_{n_1}$ hay almacenado un mensaje con la siguiente estructura: $(s_z.ta_{n_1}, (x, t, nu), (n_1, s_z.ta_{n_1}) \ldots (n_j, t_j) \ldots (n_l, t_l))$, siendo $x = s_z.sim\_id_{n_j}.id$, $t = s_z.sim\_id_{n_j}.ta$ y $nu \leq s_z.sim\_id_{n_j}.nu$. A partir de este mensaje, el nodo n_1 llegará a conocer al candidato mayor que le sucede, n_j.

Según la propiedad B.2.20, $s_{z_n}.cand\_succ_{n_1} = s_{z_n}.st\_avsrsp_{n_1}.sid$. Sustituyendo el valor que contiene $s_{z\prime\prime}.st\_avsrsp_{n_1}.sid$ (propiedad B.2.72) o el que ha guardado de un mensaje *AVSRSP* reconvertido de un mensaje *AVS*, se obtiene que $s_{z\prime\prime}.cand\_succ_{n_1} = (s_{z\prime\prime}.sim\_id_{n_j}.id, s_{z\prime\prime}.sim\_id_{n_j}.ta, nu)$, siendo $nu \leq s_{z\prime\prime}.sim\_id_{n_j}.nu$. Esto implica el nodo n_1 conoce la identidad del candidato sucesor que aparece en el camino *wait* al que pertenece. El nodo sucesor de n_1 se comporta con la identidad simulada del nodo n_j o con parte de ella.

$s_z.sim\_id_{n_j} < s_z.sim\_id_{n_1}$:

De acuerdo con la propiedad B.2.73, habrá un mensaje *AVS* en el canal dirigido a $x = last(s_{z_n}.st\_alg_{n_1}.path).id$. Considerando la propiedad de equidad débil sobre la partición y que ninguna de las acciones del algoritmo puede deshabilitar la acción $rcvAVS_x(n_1, m)$, finalmente se ejecutará. El mensaje *AVS* que se almacenará en $s_{z_n}.set\_st\_avs_x$ contiene los siguientes campos: $(s_{z_n}.sim\_id_{n_1}, last(s_{z_n}.st\_alg_{n_1}.path).ta, path, false)$, donde $path = (n_1, s_{z_n}.ta_{n_1}) \cdot \gamma_1 \cdot (n_j, t_j) \cdot \gamma_2$. Hay que recordar que la ruta que aparece en el antecedente de la propiedad, $p = (n_1, s_{z_n}.ta_{n_1}) \cdot \gamma_1 \cdot (n_j, t_j)$, puede coincidir total o parcialmente con la ruta del mensaje *AVS*.

A continuación se comprueba que ninguna acción del algoritmo deshabilita a la acción $rcvAVS_x(n_1,\ m)$:

- $\pi_z \in \{StartAddArc_x(y)\colon\ y \in \mathcal{N}\}$ o $\pi_z \in \{EndAddArc_x(y,\ t)\colon\ y \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_z \in \{StartDelArc_x(y,\ t)\colon\ y \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_z = initiate_x$ o $\pi_z \in \{rcvALG_x(y,\ m)\colon y \in \mathcal{N} \wedge m \in M_{ALG}\}$ o $\pi_z = firstAVS_x$ o $\pi_z \in \{rcvAVSRSP_x(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$ o $\pi_z = sndAVS_x$ o $\pi_z = sndAVSRSP_x$ o $\pi_z \in \{dltINF_x(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{INF}\}$. Todas estas acciones están habilitadas porque$status.id_x = known$. Sin embargo, sus efectos no retiran mensajes *AVS* del canal, no modifican ninguno de los campos del mensaje *AVS* dirigido al nodo x, ni cambian su tiempo de activación. Alguna de estas acciones puede generar mensajes *AVS*, pero su destino nunca puede ser el mismo nodo x. Todo esto implica que tras la ejecución de cualquiera de estas acciones, la acción $rcvAVS_x(n_1,\ m)$ seguiría habilitada.

- $\pi_z \in \{EndDelArc_x(y)\colon\ y \in \mathcal{N}\}$ o $\pi_z = Abort_x$. Estas acciones incrementan el tiempo de activación del nodo x. La propiedad B.2.1 indica que el tiempo asociado al nodo x en la ruta del mensaje *AVS* cumple que $t_x \leq s.ta_x$. Después de ejecutar cualquiera de estas acciones, es evidente que $t_x < s\prime.ta_x$ y se siguen verificando las condiciones de habilitación de $rcvAVS_x(n_1,\ m)$. La acción $Abort_x$ puede generar mensajes *AVS*, pero ninguno con destino el propio nodo x.

- $\pi_z \in \{rcvINF_x(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{INF}\}$. La acción está habilitada cuando $status.id_x = unknown$ y, por tanto, no se puede ejecutar.

Sabiendo que el mensaje *AVS* finalmente será recibido, se van a analizar distintas evoluciones del sistema según qué nodo sea su destino:

- $x = last(s_{z_n}.st\_alg_{n_1}.path).id = n_j$. En esta situación $path = p = (n_1,\ s_{z_n}.ta_{n_1})\cdot\gamma_1\cdot(n_j,\ t_j)$. Como $m_{AVS}.fwd = false$, $s_{z_1}.status.alg_{n_j} = candidate$ ($s_{z_1}.status.id_{n_j} = known$), la acción $rcvAVS_{n_j}(n_1,\ m)$ se ejecuta y el mensaje *AVS* se almacena en $s_{z_1}.set\_st\_avs_{n_j}$. Este efecto se produce porque $penult(m.path) \in s_{z_1}.setPred_{n_j}$ y $s_{z_1}.sim\_id_{n_j} < m_{AVS}.sid = s_{z_1}.sim\_id_{n_1}$. Como la ruta del mensaje verifica $wait(n_1,\ n_j,\ p,\ s_{z_1})$, se puede asegurar que el nodo $penult(m.path).id$ está contenido en el conjunto de predecesores del nodo n_j

- $x = last(s_{z_n}.st\_alg_{n_1}.path).id$ y $x \neq n_j$:

  1. $status.id_x = unknown$: la acción $rcvAVS_x(n_1,\ m)$ no está habilitada y el mensaje *AVS* permanece en el canal hasta que se verifiquen las precondiciones de la acción. Para que el nodo x proceda a la recepción del mensaje

AVS es necesario que éste reciba un mensaje *INF* que convierta su estado en *known*.

Dado que el mensaje *AVS* es el efecto de la acción $firstAVS_{n_1}$, el primer nodo de su ruta coincide con el nodo n_1. Además, se sabe que tanto el nodo x como el nodo n_1 eran *candidate* y los nodos entre ellos, *dummy*. En el caso base de la inducción $L = 2$, la ruta p cumple $wait(n_1, n_j, p, s_z)$, siendo los nodos n_1 y n_j candidatos. Esto implica que los nodos existentes entre el nodo x y el nodo n_j se tienen que activar para alcanzar esos requisitos.

A través de sendos mensajes *INF*, el nodo n_j llegará a adquirir la identidad simulada del nodo x. La formación de un mensaje *AVS* precisa que $sim\_id_{n_1} > sim\_id_x$ y, en el caso que se estudia, se debe verificar que $sim\_id_{n_j} < sim\_id_{n_1}$. Esto supone que, para que se cumplan todas las relaciones, el nodo n_j adquirirá una identidad simulada superior a sim_id_x. A partir de ese instante, $status.id_x = known$ y el mensaje *AVS* se podrá almacenar en $set\_st\_avs_x$ siempre que $penult(m.path) \in setPred_x$ (opción 2). Si se hubiera ejecutado la acción $StartDelArc_x(n\prime, t\prime)$), se plantearía la situación de la opción 3.

La identidad simulada que viaja entre estos nodos es sim_id_x o $sim\_id_{x\prime}$, siendo $sim\_id_{x\prime} > sim\_id_x$. Como el origen del mensaje *AVS* se debe a la relación de las identidades simuladas de nodo n_1 y del nodo x, se tiene que $sim\_id_{n_1} > sim\_id_x$. Además, como se debe verificar que $sim\_id_{n_j} < sim\_id_{n_1}$ y el nodo n_j adquirirá la identidad simulada que deje pasar el nodo x, resulta imposible que el nodo x admita una nueva identidad tal que $sim\_id_{x\prime} > sim\_id_{n_1}$. En resumen, la identidad simulada que llega hasta el nodo n_j es o bien sim_id_x ($sim\_id_{n_1} > sim\_id_x$) o bien $sim\_id_{x\prime}$, pero $sim\_id_{n_1} > sim\_id_{x\prime} > sim\_id_x$. A partir del instante en que $status.id_x = known$, el mensaje *AVS* se puede almacenar en $set\_st\_avs_x$ si $penult(m.path) \in setPred_{n_j}$.

2. $status.id_x = known \wedge penult(m.path) \in setPred_x$: la acción $rcvAVS_x(n_1, m)$ se ejecuta y el mensaje *AVS* se almacena en $set\_st\_avs_x$.

3. $status.id_x = known \wedge penult(m.path) \notin setPred_x$: el nodo x se ha activado antes de recibir el mensaje *AVS*, es decir, se han ejecutado las acciones $EndDelArc_x(n)$ y $StartDelArc_x(n\prime, t\prime)$. En esta situación, la acción $rcvAVS_x(n_1, m)$ se ejecuta y se forma un nuevo mensaje *AVS*, $m\prime$, tal que $m\prime.fwd = true$, $m\prime.sid = sim\_id_{n_1}$ y $last(m\prime.path).id = penult(m.path).id$. El destino del nuevo mensaje *AVS*, $m\prime$, será precisamente $last(m\prime.path).id$. En caso de que el mensaje *AVS* vaya dirigido a n_j, se considerará el apartado anterior en el que $x = n_j$. Si no coincide el destino con n_j, cualquiera de las opciones de este apartado serán posibles.

Resumiendo, el mensaje *AVS* se almacenará en set_st_avs del nodo destino del mensaje o se reenviará al nodo anterior en la ruta. Todo nodo de la ruta entre n_j y n_x que inicialmente fue *dummy*, necesitará ejecutar las acciones $EndDelArc_n(n\prime)$ y $rcvINF_n(n\prime,\ m)$ antes de poder almacenar el mensaje en $set\_st\_avs_n$ y sin haber ejecutado $StartDelArc_n(n\prime\prime,\ t\prime\prime)$ o reenviarlo con $fwd = true$ tras haber ejecutado $StartDelArc_n(n\prime\prime,\ t\prime\prime)$.

Seguidamente se considerarán las acciones que podrían eliminar el mensaje *AVS* almacenado de $set\_st\_avs_x$. Estas acciones son: $StartDelArc_x(n_k,\ t_k)$, $Abort_x$ y $sndAVS_x$. Aunque la acción $sndAVSRSP_x$ no retira el mensaje de $set\_st\_avs_x$, modifica el valor del campo *bool* pasándolo a *true*. A pesar de ello, este cambio no influye. Cuando se ejecuta $StartDelArc_x(n_k,\ t_k)$, siendo $status.alg_x = active$ o se ejecuta $Abort_x$ porque $status.alg_x = victim$, el mensaje de $set\_st\_avs_x$ se reconvierte en un mensaje *AVS* que verifica la propiedad. Por otra parte, la acción $sndAVS_x$ no estará habilitada si $status.alg_x = candidate$ y se ha ejecutado previamente $EndDelArc_x(n)$. En ese caso, como $st\_avsrsp_x = NULL$, la acción no está habilitada y, en consecuencia, $set\_st\_avs_x$ mantiene almacenado el mensaje. Sin embargo, la acción podrá ejecutarse si no tuvo lugar previamente la acción $EndDelArc_x(n)$. El efecto de esta acción eliminará el mensaje de $set\_st\_avs_x$, pero dirigirá un nuevo mensaje *AVS* a un nodo que cumpla las condiciones para que sea almacenado y forme parte de un camino que verifique *wait*.

Hay que recordar que las esperas que aparecen en la ruta del mensaje *AVS* entre el nodo n_j y el nodo x se irán eliminando mediante la acción combinada de $StartDelArc_n(n\prime\prime,\ t\prime\prime)$ y $EndDelArc_n(n\prime)$. Esto se debe a que el nodo n_j tiene que llegar a ser *candidate* y está incluido en la ruta del mensaje *AVS* habiendo sido inicialmente *dummy*. Si el nodo que ejecuta la acción *StartDelArc* ya ha almacenado el mensaje *AVS* en su set_st_avs, los efectos de esta acción pondrán en marcha el proceso para reenviar su contenido en forma de mensaje *AVS* al nodo que aparece como *penult* de la ruta. Este procedimiento es equivalente al que tiene lugar en la rececpción de un mensaje *AVS* en el que la ruta no se corresponde con esperas reales. En cualquier caso, el mensaje *AVS* alcanzará el nodo n_j. Después de recibir la correspondiente identidad simulada (sim_id_x), se podrá ejecutar la acción $rcvAVS_{n_j}(y,\ m)$. Los efectos de la misma permitirán que el mensaje *AVS* se almacene en $set\_st\_avs_{n_j}$ porque $penult(m.path) \in setPred_{n_j}$.

Observando la ruta almacenada, m, en $s_{z_n}.set\_st\_avs_{n_j}$ se aprecia que $first(m.path) = first(p) = (n_1,\ s_{z_n}.ta_{n_1})$. Así que, el nodo n_j cuenta con información correcta sobre el nodo *candidate* que le precede en la ruta. En la propiedad B.2.58 se establece que $m.sid = s_{z_n}.sim\_id_{n_1}$, $s_{z_n}.sim\_id_{n_1} > s_{z_n}.sim\_id_{n_j}$ y para cualquier nodo intermedio, n_l que pudiera ser *candidate*, $s_{z_n}.sim\_id_{n_1} > s_{z_n}.sim\_id_{n_l}$. Además, la existencia del mensaje *AVS* en el canal en el que $first(m.path).ta = s_z.ta_{first(m.path).id}$ asegura que $s_z.nofirstAVS_x = false$ siendo $first(m.path).id = x$ (propiedad B.2.70).

En conclusión, la propiedad se verifica en el caso inicial de $L = 2$ (hay dos nodos candidatos en la ruta que cumple la definición de *wait*). Si $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$, el nodo n_1 conoce a n_j, el candidato que le sucede en la ruta considerada y cuya identidad simulada es superior a la que él posee él. Sin embargo, cuando $s_z.sim\_id_{n_1} > s_z.sim\_id_{n_j}$, el nodo n_j conoce la identidad simulada del candidato que le precede y es mayor que él, n_1.

Caso inductivo $L = N$:

Asumiendo que la propiedad se cumple para $L = N - 1$ (hipótesis inductiva), se va a comprobar que la propiedad se verifica igualmente cuando hay un número indeterminado de nodos en estado *candidate* en una ruta p que cumple $wait(n_1, n_j, p, s_z)$.

Para empezar la demostración, se asume que en la ruta p debe existir un nodo n_k que cuenta con la identidad simulada mayor sin considerar al nodo n_1 y n_j, esto es, si $L > 2$ y $\exists\ z_0$ tal que $\forall\ z \geq z_0$ en el que se verifica que $min(s_z.sim\_id_{n_1}, s_z.sim\_id_{n_j}) > max(\{s_z.sim\_id_{n_l}: 1 < l < j,\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\})$ entonces $\exists\ n_k = \max(\{s_z.sim\_id_{n_l}: 1 < l < j, n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\})$.

A continuación, se procederá a dividir en dos partes la ruta p a través de ese nodo n_k y se supondrá que en cada una de esas partes se verifica la propiedad, esto es, se verifica que $\exists\ z_0$: $\forall\ z \geq z_0$ tal que:

- $wait(n_k, n_j, p\prime, s_z) \wedge s_z.status.alg_{n_k} = s_z.status.alg_{n_j} = candidate \wedge \forall\ n \in nodes(p\prime),\ s_z.status.alg_n \neq victim \wedge s_z.sim\_id_{n_j} > s_z.sim\_id_{n_k} > max(\{s_z.sim\_id_{n_l}: k < l < j,\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\}) \wedge || \{n_l \in nodes(p\prime): s_z.status\_alg_{n_l} = candidate\} || \leq N - 1$, siendo $k \leq l \leq j$

- $wait(n_1, n_k, p\prime\prime, s_z) \wedge s_z.status.alg_{n_1} = s_z.status.alg_{n_k} = candidate \wedge \forall\ n \in nodes(p\prime\prime),\ s_z.status.alg_n \neq victim \wedge s_z.sim\_id_{n_1} > s_z.sim\_id_{n_k} > max(\{s_z.sim\_id_{n_l}: 1 < l < k,\ n_l \in nodes(p) \wedge s_z.status.alg_{n_l} = candidate\} \wedge || \{n_l \in nodes(p\prime\prime): s_z.status\_alg_{n_l} = candidate\} || \leq N - 1$, siendo $1 \leq l \leq k$

Por lo tanto, para cada uno de los trozos de la ruta p, según la hipótesis inductiva, $\exists\ z_1 \geq z_0$ tal que $\forall\ z_2 \geq z_1$ se verifica que:

- $s_{z_2}.cand\_succ_{n_k} = (s_{z_2}.sim\_id_{n_j}.id,\ s_{z_2}.sim\_id_{n_j}.ta,\ nu)$, siendo $nu \leq s_{z_2}.sim\_id_{n_j}.nu \wedge$

- $(s_{z_2}.sim\_id_{n_1},\ t,\ p,\ false) \in s_{z_2}.set\_st\_avs_{n_k}$ tal que $t_1 = s_{z_2}.ta_{n_1} \wedge t \in \mathbb{N}$.

Así que, de acuerdo a la hipótesis inductiva, el nodo n_k tiene almacenado un mensaje, m, en $s_{z_2}.set\_st\_avs_{n_k}$ tal que $m.bool = false$ y, además,$s_{z_2}.st\_avsrsp_{n_k} \neq NULL$. Esto implica que el nodo n_k cuenta con información correcta relacionada

con la mayor identidad simulada del nodo *candidate* que le precede y que le sucede respectivamente. Considerando que $\forall\ z_2 \geq z_1$ la información correspondiente a n_1 y a n_j no varía y la acción $sndAVS_{n_k}$ o $sndAVSRSP_{n_k}$ queda habilitada. Para evitar el envío de mensajes redundantes y/o repetidos, una vez que se proceda a combinar los almacenes mencionados, se eliminará m de $set\_st\_avs_{n_k}$ (acción $sndAVS_{n_k}$) o se cambiará $m.bool$ a *true* (acción $sndAVSRSP_{n_k}$).

Por otra parte, como $\forall\ z \geq z_0$ la ruta p cumple la definición de camino $wait(n_1, n_j, p, s_z)$, resulta que $s_z.blocker_{n_1} \neq NULL$ y $s_z.blocker_{n_j} \neq NULL$. Dado que $z_0 \leq z_1 \leq z_2$, la información tanto del nodo n_1 como del nodo n_j se mantiene donde la ruta p esté almacenada en s_{z_2}.

Seguidamente se va a comprobar que ninguna acción del algoritmo puede deshabilitar la acción $sndAVS_{n_k}$ o $sndAVSRSP_{n_k}$:

- $\pi_{z_3} \in \{StartAddArc_{n_k}(y)\colon y \in \mathcal{N}\}$ o $\pi_{z_3} = Abort_{n_k}$ o $\pi_{z_3} = initiate_{n_k}$ o $\pi_{z_3} \in \{rcvINF_{n_k}(y, m)\colon y \in \mathcal{N} \wedge m \in M_{INF}\}$. Cualquiera de estas acciones no está habilitada porque o bien $s_{z_3-1}.status.alg_{n_k} \neq candidate$, o bien $s_{z_3-1}.status.id_{n_k} \neq known$.

- $\pi_{z_3} \in \{EndAddArc_{n_k}(y, t)\colon y \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_{z_3} = \{dltINF_{n_k}(y, m)\colon y \in \mathcal{N} \wedge m \in M_{INF}\}$. Los efectos de estas acciones no modifican las variables que habilitan la ejecución de tanto $sndAVS_{n_k}$ como $sndAVSRSP_{n_k}$,

- $\pi_{z_3} \in \{StartDelArc_{n_k}(y, t)\colon \{n_k, y\} \subseteq \mathcal{N} \wedge t \in \mathbb{N}\}$. Esta acción no está habilitada porque o bien $s_{z_3-1}.status.alg_{n_k} \neq candidate$, o bien si $(y,\ t) \notin s_{z_3-1}.setPred_{n_k}$ se incumple la definición de camino *wait* para la ruta p considerada.

- $\pi_{z_3} \in \{EndDelArc_{n_k}(y)\colon \{n_k,\ y\} \subseteq \mathcal{N}\}$. Si se ejecuta esta acción, $(y,\ t) \notin s_{z_3-1}.setPred_{n_k}$. Esto implica que la ruta p no podría cumplir la definción de camino *wait* porque hay una espera rota.

- $\pi_{z_3} \in \{rcvALG_{n_k}(y,\ m)\colon y \in \mathcal{N} \wedge m \in M_{ALG}\}$. Si el efecto de la acción consiste solamente en retirar el mensaje ALG del canal, las acciones $sndAVS_{n_k}$ y $sndAVSRSP_{n_k}$ siguen habilitadas. Sin embargo, los efectos de la acción que precisan que $s_{z_3-1}.status.alg_{n_k} \neq candidate$ no se pueden producir. Por último, si $s_{z_3-1}.status.alg_{n_k} = candidate$, la propiedad B.2.71 establece que $s_{z_3-1}.st\_alg_{n_k} = NULL$. Como las acciones requieren que $s_{z_3-1}.st\_avsrsp_{n_k} \neq NULL$, resulta que $s_{z_3-1}.st\_alg_{n_k} \neq NULL$ ya que la aplicación de la propiedad B.2.33 así lo señala.

- $\pi_{z_3} = firstAVS_{n_k}$. De acuerdo con la propiedad B.2.51, esta acción requiere que $s_{z_3-1}.st\_avsrsp_{n_k} = NULL$ y en las acciones analizadas $s_{z_3-1}.st\_avsrsp_{n_k} \neq NULL$. Por tanto, la acción no está habilitada.

- $\pi_{z_3} \in \{rcvAVS_{n_k}(y,\, m)\colon y \in \mathcal{N} \wedge m \in M_{AVS}\}$. De los posibles efectos de esta acción, sólo el que almacena una nueva ruta en $s_{z_3}.set\_st\_avs_{n_k}$ y el que convierte $s_{z_3}.status.alg_{n_k} = victim$ pueden cambiar las precondiciones de las acciones estudiadas. La propiedad B.2.58 asegura que el mensaje *AVS* del canal no puede tener el primer nodo de la ruta en común con ninguna ruta almacenada en $s_{z_3-1}.set\_st\_avs_{n_k}$. Eso quiere decir que su almacenamiento no impide la ejecución de las acciones $sndAVS_{n_k}$ y $sndAVSRSP_{n_k}$. Finalmente, se deduce que la situación en la que $s_{z_3}.status.alg_{n_k}$ pasa a ser *victim* es imposible porque $z_3 < z$ y un nodo *victim* no puede convertirse de nuevo en *candidate* como exige el antecedente de la propiedad ($s_z.status.alg_{n_k} = candidate$).

- $\pi_{z_3} \in \{rcvAVSRSP_{n_k}(y,\, m)\colon y \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$. Los efectos de esta acción pueden almacenar una nueva ruta en $s_{z_3}.st\_avsrsp_{n_k}$ y adquirir un nuevo valor en $s_{z_3}.cand\_succ_{n_k}$. Considerando que dicha ruta cumple la propiedad B.2.58, se asegura que en la variable $cand\_succ_{n_k}$ se almacena la identidad del mayor sucesor que sigue siendo el mismo nodo n_i gporque $m.ta = s_{z_3-1}.ta_{n_k}$. En conclusión, los efectos de esta acción no afectan a la ejecución de las acciones consideradas.

Teniendo en cuenta la propiedad de equidad débil para las clases de la partición, se puede asegurar que en algún momento el nodo n_k ejecutará la acción $sndAVS_{n_k}$ o $sndAVSRSP_{n_k}$:

- Si, en el estado s_{z_1}, se cumple que $\exists\ m \in s_{z_1}.set\_st\_avs_{n_k}$ tal que $m.bool = false$, entonces la clase $\{sndAVS_{n_k},\ sndAVSRSP_{n_k}\}$ está habilitada. De este modo, dado que $\forall\ z_2 \geq z_1$ se verifica que $\exists\ m \in s_{z_2}.set\_st\_avs_{n_k}$ tal que $m.bool = false$, las únicas acciones que deshabilitan dicha clase, manteniendo la información relativa a los nodos n_1 y n_j, son esas mismas acciones. Por lo tanto, la asunción de equidad sobre la partición establece que una de las acciones finalmente se ejecuta, es decir, $\exists\ z_3 > z_1$ tal que: $\pi_{z_3} \in \{sndAVS_{n_k},\ sndAVSRSP_{n_k}\}$, $\exists\ m \in s_{z_3-1}.set\_st\_avs_{n_k}$ tal que $m.bool = false$ y $\exists\ m \in s_{z_4}.set\_st\_avs_{n_k}$ tal que $m.bool = true$ o $\nexists\ m \in s_{z_4}.set\_st\_avs_{n_k}$, $\forall\ z_4 \geq z_3$.

- Si, en el estado s_{z_1}, se cumple que $\exists\ m \in s_{z_1}.set\_st\_avs_{n_k}$ tal que $m.bool = true \vee \nexists\ m \in s_{z_1}.set\_st\_avs_{n_k}$, entonces la acción $sndAVS_{n_k}$ o $sndAVSRSP_{n_k}$ se ha ejecutado previamente, o lo que es lo mismo, $\exists\ z_3 \leq z_1$ tal que $\exists\ m \in s_{z_3-1}.set\_st\_avs_{n_k}$ tal que $m.bool = false$, $\pi_{z_3} = \{sndAVS_{n_k},\ sndAVSRSP_{n_k}\}$ y $\exists\ m \in s_{z_4}.set\_st\_avs_{n_k}$ tal que $m.bool = true \vee \nexists\ m \in s_{z_4}.set\_st\_avs_{n_k}$, $\forall\ z_4 \geq z_3$.

En ambos casos, se cumplen ciertas propiedades $\forall\ z_4 \geq z_3$:

- Dado que $\forall\ z_4 \geq z_3$ se verifica que $\exists\ m \in s_{z_4}.set\_st\_avs_{n_k}$ tal que $m.bool = true \vee \nexists\ m \in s_{z_4}.set\_st\_avs_{n_k}$, el nodo n_1 será predecesor del nodo n_k en s_{z_4} y el nodo

n_k sucesor de n_1 en s_{z_4}. Como los tiempos de activación se pueden incrementar, $s_{z_1}.ta_{n_1} \leq s_{z_4}.ta_{n_1} \wedge s_{z_1}.ta_{n_j} \leq s_{z_4}.ta_{n_j}$. Por otro lado, como $z_1 \geq z_0$ y $\forall\ z \geq z_0$ se verifica que $s_z.blocker_{n_1} \neq NULL \wedge s_z.blocker_{n_j} \neq NULL$. Además, $s_z.ta_{n_1} = s_{z_1}.ta_{n_1} \wedge s_z.ta_{n_j} = s_{z_1}.ta_{n_j}$. De la misma forma, como $\forall\ z_4 \geq z_3$, se verifica que $s_{z_1}.ta_{n_1} \leq s_{z_4}.ta_{n_1} \wedge s_{z_1}.ta_{n_j} \leq s_{z_4}.ta_{n_j}$ y $\forall\ z \geq z_0$ se cumple que $s_z.ta_{n_1} = s_{z_1}.ta_{n_1} \wedge s_z.ta_{n_j} = s_{z_1}.ta_{n_j}$, se verificará que $s_{z_4}.ta_{n_1} = s_{z_1}.ta_{n_1} \wedge s_{z_4}.ta_{n_j} = s_{z_1}.ta_{n_j}$

- Como $\forall\ z_4 \geq z_3$ se verifica que n_1 es el predecesor del nodo n_k en s_{z_4} y n_j es el sucesor del nodo n_k en s_{z_4} y $s_{z_4}.ta_{n_1} = s_{z_1}.ta_{n_1} \wedge s_{z_4}.ta_{n_j} = s_{z_1}.ta_{n_j}$, la información del nodo n_1 y el nodo n_j se mantiene constante en s_{z_4}. Por la propiedad B.1.1, $s_{z_4}.state_{n_1} \neq aborted \wedge s_{z_4}.state_{n_j} \neq aborted$, ya que $s_{z_4}.blocker_{n_1} \neq NULL \wedge s_{z_4}.blocker_{n_j} \neq NULL$.

Finalmente, comprobado que se ejecuta la acción $\pi_{z_3} \in \{sndAVS_{n_k}, sndAVSRSP_{n_k}\}$ y confirmado todo los anterior, se demuestra que el consecuente de la propiedad es cierto para $wait(n_1,\ n_j,\ p,\ s_z)$.

- Al ejecutarse π_{z_3}, como $s_z.sim\_id_{n_j} > s_z.sim\_id_{n_1}$, el nodo n_k envía un mensaje *AVSRSP* al nodo n_1. En estados posteriores, como $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.ta_{n_1} = s_{z_1}.ta_{n_1}$ el mensaje sigue conteniendo información relevante. Este mensaje llega a ser el primero en el canal. La acción $rcvAVSRSP_{n_1}(n_k,\ m)$ se ejecuta y el nodo n_j pasa a ser $cand\_succ_{n_1}$, $\exists\ z_m > z_3$ tal que $\pi_{z_m} = rcvAVSRSP_{n_1}(n_k,\ m) \wedge s_{z_m}.cand\_succ_{n_1} = n_j$.

  Por lo tanto,

  - $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.blocker_{n_1} \neq NULL$, $s_{z_m}.cand\_succ_{n_1} = n_j$ y $z_3 < z_m$, $\forall\ z_n \geq z_m$ se verifica que $s_{z_n}.cand\_succ_{n_1} = n_j$.
  - $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.ta_{n_j} = s_{z_1}.ta_{n_j}$ y $z_3 < z_m$, $\forall\ z_n \geq z_m$ se verifica que n_j en s_{z_n} presenta las mismas características que en s_{z_1}.

  Se concluye que $\exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se verifica que $s_{z_n}.cand\_succ_{n_1} = (s_{z_n}.sim\_id_{n_j}.id,\ s_{z''}.sim\_id_{n_j}.ta,\ nu)$, siendo $nu \leq s_{z_n}.sim\_id_{n_j}.nu$ (n_1 conoce la identidad simulada mayor del camino wait considerado que pertenece al sucesor *candidate* n_j).

- Al ejecutarse π_{z_3}, como $s_z.sim\_id_{n_j} < s_z.sim\_id_{n_1}$, el nodo n_k envía un mensaje *AVS* al nodo n_j. En estados posteriores, como $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.ta_{n_j} = s_{z_1}.ta_{n_j}$ el mensaje sigue conteniendo información relevante. Este mensaje llega a ser el primero en el canal. La acción $rcvAVS_{n_j}(n_k,\ m)$ se ejecuta y el nodo n_1 pasa a ser el predecesor de n_j, $\exists\ z_m > z_3$ tal que $\pi_{z_m} = rcvAVS_{n_j}(n_k,\ m)$ y el nodo n_1 pasa a ser el predecesor de n_j.

Por lo tanto,

- $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.blocker_{n_j} \neq NULL \wedge s_{z_4}.state_{n_1} \neq aborted$, n_1 es predecesor de n_j en s_{z_m} y $z_3 < z_m$, $\forall\ z_n \geq z_m$ se verifica que n_1 es predecesor de n_j en s_{z_n}.
- $\forall\ z_4 \geq z_3$ se verifica que $s_{z_4}.ta_{n_1} = s_{z_1}.ta_{n_1}$ y $z_3 < z_m$, $\forall\ z_n \geq z_m$ se verifica que n_1 en s_{z_n} presenta las mismas características que en s_{z_1}.

En conclusión, $\exists\ z_m \geq z_0$ tal que $\forall\ z_n \geq z_m$ se verifica que $\exists\ m \in s_{z_n}.set\_st\_avs_{n_j}$ tal que $m.path = p \wedge first(m.path) = (n_1,\ s_{z_n}.ta_{n_1}) \wedge m.bool = false \wedge m.sid = s_{z_n}.sim\_id_{n_1}$ (n_j conoce la identidad simulada mayor correspondiente a un predecesor que es *candidate* y no es inmediato en el camino wait considerado, n_1). En la propiedad B.2.58 se establece que $m.sid = s_{z_n}.sim\_id_{n_1} \wedge s_{z_n}.sim\_id_{n_1} > s_{z_n}.sim\_id_{n_j} \wedge \forall\ l,\ 1 < l < i$ tal que $s_{z_n}.status.alg_{n_l} = candidate$ se tiene que $s_{z_n}.sim\_id_{n_1} > s_{z_n}.sim\_id_{n_l}$.

■

En la siguiente propiedad se asegura que el nodo elegido para resolver la situación de interbloqueo es el de mayor identidad simulada del ciclo.

Propiedad B.2.76. Sea $\alpha = s_0\ \pi_1\ s_1 \ldots \pi_z\ s_z \ldots$ una ejecución de S. $\forall\ n_i \in \mathcal{N}$, se verifica que $\exists\ p \in P^+$, $\exists\ z_0$: $\forall\ z \geq z_0$ se cumple $C(p,\ s_z) \wedge s_z.sim\_id_{n_i} = max(\{s_z.sim\_id_n$: $n \in nodes(p) \wedge s_z.status.alg_n = candidate\}) \Rightarrow \exists\ z_m \geq z_0$: $\forall\ z_n \geq z_m$, $s_{z_n}.status.alg_{n_i} = victim$.

Demostración: Se procede a demostrar esta propiedad según el número de nodos *candidate* de la ruta que conforma el ciclo. Si $||\{n \in nodes(p)$: $s_z.status.alg_n = candidate\}|| = 1$, la propiedad B.2.68 concluye. En cambio, si $||\{n \in nodes(p)$: $s_z.status.alg_n = candidate\}|| \geq 2$, se debe estudiar la evolución del sistema para comprobar cómo se va reduciendo el conjunto de candidatos del ciclo de entre los cuales se puede elegir el que pasará a víctima.

Sabiendo que la ruta p constituye un ciclo, $C(p,\ s_z)$ y, por tanto, cumple $wait(n_1,\ n_1,\ p,\ s_z)$, se puede afirmar que existe un nodo candidato con la mayor identidad simulada. Suponiendo que ese nodo es n_1, $s_z.sim\_id_{n_1} = max(s_z.sim\_id_n$: $n \in nodes(p) \wedge s_z.status.alg_n = candidate\})$. Es obvio que, sin considerar el nodo n_1, existirá otro nodo candidato, n_k, que tenga la mayor identidad simulada de los nodos restantes de la ruta p, o sea, $\exists\ n_k$ tal que $s_z.sim\_id_{n_k} = max(s_z.sim\_id_n$: $n \in nodes(p) \setminus \{n_1\} \wedge s_z.status.alg_n = candidate\})$. Esto significa que la ruta p del ciclo C se puede dividir en dos partes o subrutas y, en cada una de ellas, es aplicable la propiedad B.2.75.

Como $\forall\ z \geq z_0$ se verifica que:

- $wait(n_k,\ n_1,\ p\prime,\ s_z) \wedge s_z.status.alg_{n_k} = s_z.status.alg_{n_1} = candidate \wedge s_z.sim\_id_{n_1} > s_z.sim\_id_{n_k} > max(\{s_z.sim\_id_n\colon\ n \in nodes(p\prime) \setminus \{n_k,\ n_1\} \wedge s_z.status.alg_n = candidate\}) \wedge$

- $wait(n_1,\ n_k,\ p\prime\prime,\ s_z) \wedge s_z.status.alg_{n_1} = s_z.status.alg_{n_k} = candidate \wedge s_z.sim\_id_{n_1} > s_z.sim\_id_{n_k} > max(\{s_z.sim\_id_n\colon\ n \in nodes(p\prime\prime) \setminus \{n_1,\ n_k\} \wedge s_z.status.alg_n = candidate\})$

La propiedad B.2.75 permite asegurar que $\exists\ z_1 \geq z_0$ tal que $\forall\ z_2 \geq z_1$ se verifica que:

- $s_{z_2}.cand\_succ_{n_k} = (s_{z_2}.sim\_id_{n_1}.id,\ s_{z_2}.sim\_id_{n_1}.ta,\ nu)$, siendo

  $nu \leq s_{z_2}.sim\_id_{n_1}.nu \wedge$

- $(s_{z_2}.sim\_id_{n_1},\ t,\ p\prime\prime) \in s_{z_2}.set\_st\_avs_{n_k} \wedge first(p\prime\prime) = (n_1,\ s_{z_2}.ta_{n_1})$

En todo estado s_{z_2} posterior a s_{z_1}, el nodo n_k conoce la identidad simulada del nodo n_i. Esta información no varía de z_1 a z_2 porque el ciclo se mantiene. En algún momento el nodo n_k procesará la información que posee. Como $s_{z_2}.cand\_succ_{n_k} \leq n.sid = s_{z_2}.sim\_id_{n_1}$, siendo n un mensaje almacenado en $s_{z_2}.set\_st\_avs_{n_k}$ con $first(n.path) = (n_1,\ s_{z_2}.ta_{n_1})$, la acción $\pi_{z_3} = sndAVS_{n_k}$ estará habilitada. La ejecución de esta acción conlleva el envío de un mensaje AVS a un nodo x que puede coincidir o no con el nodo n_1.

Seguidamente, se comprueba que ninguna acción del algoritmo puede deshabilitar la acción $sndAVS_{n_k}$ a excepción de ella misma:

- $\pi_{z_3} \in \{StartAddArc_{n_k}(y)\colon\ y \in \mathcal{N}\}$ o $\pi_{z_3} = Abort_{n_k}$ o $\pi_{z_3} = initiate_{n_k}$ o $\pi_{z_3} \in \{rcvINF_{n_k}(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{INF}\}$. Para que cualquiera de estas acciones se ejecute es preciso que $s_{z_3-1}.status.alg_{n_k} \neq candidate$. Así que, ninguna de ellas está habilitada.

- $\pi_{z_3} \in \{EndAddArc_{n_k}(y,\ t)\colon\ y \in \mathcal{N} \wedge t \in \mathbb{N}\}$ o $\pi_{z_3} \in \{dltINF_{n_k}(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{INF}\}$. Si se ejecuta cualquiera de estas acciones, la acción $sndAVS_{n_k}$ sigue estando habilitada.

- $\pi_{z_3} \in \{StartDelArc_{n_k}(y,\ t)\colon\ y \in \mathcal{N} \wedge t \in \mathbb{N}\}$. Si se ejecuta esta acción o bien $s_{z_3-1}.status.alg_{n_k} \neq candidate$, o bien $s_{z_3-1}.status.alg_y = aborted$. Es obvio que con la primera condición la acción no está habilitada. En cambio, la segunda condición implica que el nodo y no forma parte de una ruta que pudiera cumplir la aserción *wait*. Así que, los efectos de la acción en este caso no afectan a la acción $sndAVS_{n_k}$.

- $\pi_{z_3} \in \{EndDelArc_{n_k}(y)\colon\ y \in \mathcal{N}\}$. Cuando se ejecuta esta acción, el nodo y no está incluido en la ruta que cumple la aserción *wait* del enunciado de la propiedad porque $(y,\ t) \notin s_{z_3-1}.setPred_{n_k}$. Esto significa que los efectos de esta acción no afectan.

- $\pi_{z_3} \in \{rcvALG_{n_k}(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{ALG}\}$. Si el efecto de la acción consiste únicamente en retirar el mensaje ALG del canal, la acción $sndAVS_{n_k}$ sigue habilitada. Para el resto de efectos la acción no está habilitada. En unos casos porque se requiere que $s_{z_3-1}.status.alg_{n_k} \neq candidate$ y en otros, siendo $s_{z_3-1}.status.alg_{n_k} = candidate$ porque la propiedad B.2.71 establece que $s_{z_3-1}.st\_alg_{n_k} = NULL$. Como la precondición de la acción $sndAVS_{n_k}$ precisa $s_{z_3-1}.st\_avsrsp_{n_k} \neq NULL$ y la propiedad B.2.33 asegura que entonces $s_{z_3-1}.st\_alg_{n_k} \neq NULL$, se concluye que tampoco puede estar habilitada.

- $\pi_{z_3} = firstAVS_{n_k}$. De acuerdo con la propiedad B.2.51, las precondiciones de esta acción implican que $s_{z_3-1}.st\_avsrsp_{n_k} = NULL$. Como la acción $sndAVS_{n_k}$ exige que $s_{z_3-1}.st\_avsrsp_{n_k} \neq NULL$, se concluye que la acción $firstAVS_{n_k}$ no puede ejecutarse.

- $\pi_{z_3} \in \{rcvAVS_{n_k}(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{AVS}\}$. De los posibles efectos de esta acción, sólo el que almacena una nueva ruta en $s_{z_3}.set\_st\_avs_{n_k}$ y el que convierte $s_{z_3}.status.alg_{n_k} = victim$ pueden cambiar las precondiciones de la acción estudiada. Por un lado, la propiedad B.2.58 asegura que el mensaje AVS del canal no puede tener el primer nodo de la ruta en común con ninguna ruta almacenada en $s_{z_3-1}.set\_st\_avs_{n_k}$. Eso quiere decir que su almacenamiento no impediría la ejecución de la acción $sndAVS_{n_k}$. En el caso de que el efecto de la acción sea que $s_{z_3}.status.alg_{n_k} = victim$, se contradice el antecedente de la propiedad que asume que $s_z.status.alg_{n_k} = candidate$. Como $z_3 < z$ y un nodo *victim* no puede convertirse de nuevo en *candidate*, la situación es imposible.

- $\pi_{z_3} \in \{rcvAVSRSP_{n_k}(y,\ m)\colon\ y \in \mathcal{N} \wedge m \in M_{AVSRSP}\}$. Los efectos de esta acción pueden llevar a almacenar una nueva ruta en $s_{z_3}.st\_avsrsp_{n_k}$ y un nuevo valor $s_{z_3}.cand\_succ_{n_k}$. Considerando que dicha ruta cumple la propiedad B.2.58, se asegura que en la variable $cand\_succ_{n_k}$ se almacena la identidad del mayor sucesor que sigue siendo el mismo nodo porque $m.ta = s_{z_3-1}.ta_{n_k}$. En conclusión, los efectos de esta acción no afectan a la acción $sndAVS_{n_k}$.

- $\pi_{z_3} = sndAVSRSP_{n_k}$. La relación entre $s_{z_3-1}.cand\_succ_{n_k}$ y el campo *sid* del mensaje almacenado en $s_{z_3-1}.set\_st\_avs_{n_k}$ impide que se ejecute esta acción.

A partir de ese instante de ejecución, $\forall\ z_4 \geq z_3$ se cumple que el nodo n_k conoce a su sucesor y a su predecesor que es precisamente el nodo n_1 y $s_{z_1}.ta_{n_1} \leq s_{z_4}.ta_{n_1}$.

Por otra parte, como $z_1 \geq z_0$ y $\forall\ z \geq z_0$ se verifica que las esperas del ciclo no varían, entonces $s_z.ta_{n_1} = s_{z_1}.ta_{n_1}$. De este modo, dado que $\forall\ z_4 \geq z_3$ se verifica que $s_{z_1}.ta_{n_1} \leq s_{z_4}.ta_{n_1}$ y $\forall\ z \geq z_0$ se cumple que $s_z.ta_{n_1} = s_{z_1}.ta_{n_1}$, se confirma que $s_{z_4}.ta_{n_1} = s_{z_1}.ta_{n_1}$.

Al ejecutarse finalmente la acción $\pi_{z_3} = sndAVS_{n_k}$ puede suceder que el nodo destino del mensaje no coincida con n_1. En ese caso, el mensaje *AVS* es reenviado a los nodos que aparecen en la ruta, pero que ya no representan esperas reales (*fwd* = *true*). Hasta que la ruta de ese mensaje *AVS* en retroceso no coincida con la ruta correspondiente a las esperas reales que definen el ciclo, se procederá al reenvío de mensajes *AVS*. En el caso de que el destino del mensaje *AVS* fuera el nodo n_1, el campo *fwd* del mismo sería *false* y la acción $rcvAVS_{n_1}(y,\ m)$ estaría habilitada cumpliendo esa condición. Cuando la acción $rcvAVS_{n_1}(y,\ m)$ se ejecuta siendo *fwd* = *true*, se cumple la precondición que indica que el tiempo de activación del nodo n_1 es superior al que aparece asociado a él en la última posición de la ruta. Esto es fácil comprobar ya que el nodo n_1 pertenece al ciclo y aparece ya en la primera parte de la ruta p, llamada p''. Por la definición de camino *wait* se sabe que su tiempo es correcto. Sin embargo, el mensaje *AVS* se pudo formar concatenando la ruta de $s_{z_3-1}.set\_st\_avs_{n_k}$ con la de $s_{z_3-1}.st\_avsrsp_{n_k}$ y en ella el nodo n_1 puede aparecer bloqueado por un nodo y esa espera haber desaparecido antes de la formación del ciclo.

En consecuencia, existirá un estado z_m tal que $z_m > z_3$ en el que la acción $\pi_{z_m} = rcvAVS_{n_1}(y,\ m)$ sea finalmente ejecutada, consiguiendo que $s_{z_m}.status.alg_{n_1} = victim$.

Asumiendo que la ruta p del ciclo persiste en el estado s_z, siendo $z \geq z_0$, y además $s_z.status_{n_1} \neq aborted$, si el nodo n_1 alcanza el estado *victim* en s_{z_m}, $\forall\ z \geq z_0$ se tiene que $s_z.status.alg_{n_1} \neq aborted$ (propiedad B.2.60 y propiedad B.1.6), y, por tanto, $\forall\ z_n \geq z_m$ se cumple que $s_{z_n}.status.alg_{n_1} = victim$. En consecuencia, la propiedad se verifica.

■

Esta última propiedad garantiza que, si existe un ciclo, habrá un nodo del mismo que tenga habilitada la acción *Abort* para resolver el interbloqueo detectado.

Propiedad B.2.77. Sea $\alpha = s_0\ \pi_1\ s_1 \ldots \pi_z\ s_z \ldots$ una ejecución de S. $\forall\ n_i \in \mathcal{N}$, se verifica que: $\exists\ p \in P^+$ tal que $C(p,\ s_z) \Rightarrow \exists\ z_n > z_0$: $\pi_{z_n} \in \{Abort_{n_i}\}$ tal que $n_i \in nodes(p)$.

Demostración: Se va a demostrar la propiedad suponiendo que la propiedad no es cierta, es decir, se supondrá que la ruta p verifica $C(p,\ s_{z_0})$ y que el ciclo permanece sin resolver indefinidamente, $\forall\ z_n > z_0$ se verifica $C(p,\ s_{z_n})$.

Tal y como indica la propiedad B.2.62, un ciclo termina de formarse al ejecutar la acción $EndAddArc_x(y,\ t)$. Por otra parte, la propiedad B.2.66 garantiza que hay un instante en el que la variable *status.id* de todos los nodos del ciclo vale *known*. Esto

permite que si en el ciclo inicialmente no hay ningún nodo cuyo estado sea *candidate*, al menos uno llegará a ejecutar la acción $initiate_x$ (equidad débil). Si hubiera más de un nodo *candidate* entre los nodos del ciclo considerado, uno de ellos será el de mayor identidad simulada: $\exists\ n_i \in nodes(p)$ tal que $\forall\ z > z_0$ se verifica que $wait(n_i, n_i, p, s_z)$ y, además, $s_z.sim\_id_{n_i} = max(sim\_id_{nodes(p)})$.

De acuerdo con la propiedad B.2.76, el nodo n_i detectará que forma parte de un interbloqueo: $\exists\ z_1 \geq z_0$ tal que $s_{z_1}.status.alg_{n_i} = victim$. De este modo, teniendo en cuenta la propiedad B.2.61 que asegura que el estado del nodo n_i llegará a ser *aborted*, se concluye que la acción $Abort_{n_i}$, que está habilitada y es la única que produce ese cambio de estado, se ejecutará finalmente. Así que, es obvio que $\exists\ z_n > z_0$ tal que $\pi_{z_n} = Abort_{n_i}$, lo que contradice la suposición inicial de que la propiedad era falsa. ■

B.3. Equivalencias entre variables de *G* y *A*

A continuación se exponen algunas propiedades que relacionan los valores de las variables de los autómatas G y A en los estados alcanzables de S, siendo S el autómata resultante de la composición de G y A. Estas propiedades permiten reducir la extensión del código de las acciones de S. Para la demostración por inducción de las propiedades se supone una ejecución $\alpha = s_0\ \pi_1\ s_1\ \ldots\ \pi_z\ s_z\ \ldots \in execs(S)$.

La primera propiedad evidencia que, en los estados alcanzables de S, se puede obviar cualquier referencia a *t_activ* (variable del autómata G) pues su contenido coincide con el de *ta* (variable del autómata A).

Propiedad B.3.1. Sea s un estado alcanzable de S. Se verifica que $\forall\ i \in \mathcal{N}$: $s.t\_activ_i = s.ta_i$.

Demostración: En el estado inicial, s_0, se verifica que $\forall\ i \in \mathcal{N}$: $s_0.t\_activ_i = s_0.ta_i = 0$, por lo que la propiedad se cumple. Las únicas acciones que modifican las variables incluidas en la propiedad son las del conjunto $\{EndDelArc_i(j),\ Abort_i: \{i, j\} \subseteq \mathcal{N}\}$. Como en estas acciones se incrementa en una unidad los valores de las dos variables, la hipótesis inductiva concluye. ■

La siguiente propiedad indica que el algoritmo sólo considera predecesor de un nodo a otro nodo que esté esperándole de forma real. Se recuerda que los valores *sent* y *released* recogidas en los conjuntos *set_waiters* reflejan situaciones de transmisión de información entre nodos mediante mensajes. Sólo los elementos de *set_waiters* con el valor *received* reflejan que hay una espera real entre los nodos involucrados del sistema.

Propiedad B.3.2. Sea s un estado alcanzable de S. Se verifica que $\exists\ \{i,\ j\} \subseteq \mathcal{N}$, $\exists\ t \in \mathbb{N}$ tales que $(i,\ t) \in s.setPred_j \Leftrightarrow (i,\ t,\ received) \in s.set\_waiters_j$.

Demostración: En el estado inicial, s_0, se verifica que $\forall\ i \in \mathcal{N}$: $s_0.setPred_i = \emptyset$ y $s_0.set\_waiters_i = \emptyset$, por lo que se cumple la propiedad. Las acciones que afectan a las variables de la propiedad son:

- $\pi_z = StartAddArc_j(i)$. Los efectos de la acción no modifican la variable $setPred_j$. Aunque se incluyen elementos en $set\_waiters_j$, ni se incorporan ni se eliminan tuplas del tipo $(i,\ t,\ received)$. Por hipótesis inductiva se cumple la propiedad.
- $\pi_z \in \{EndAddArc_j(i,\ t)\colon\ t \in \mathbb{N}\}$. Tras ejecutarse la acción, $(i,\ t,\ received) \in s_z.set\_waiters_j$ y, además, $(i,\ t) \in s_z.setPred_j$.
- $\pi_z \in \{StartDelArc_j(i,\ t)\colon\ t \in \mathbb{N}\}$. Los efectos de esta acción eliminan $(i,\ t,\ received)$ de $s_{z-1}.set\_waiters_j$ y también, $(i,\ t)$ de $s_{z-1}.setPred_j$.
- $\pi_z = EndDelArc_i(j)$. La variable $setPred_j$ no cambia al ejecutarse la acción. En cambio, $set\_waiters_j$ puede variar eliminándose una terna: $(i,\ t\_activ_i,\ released)$ o $(i,\ t\_activ_i,\ sent)$. Si alguna de esas ternas estaba en $s_{z-1}.set\_waiters_j$, por la propiedad B.1.4, $(i,\ t\_activ_i,\ received) \notin s_{z-1}.set\_waiters_j$. Por tanto, la hipótesis inductiva junto con la propiedad B.3.1 aseguran que $(i,\ ta_i) \notin s_z.setPred_j$. Como la acción no incluye $(i,\ t\_activ_i,\ received)$ en $s_z.set\_waiters_j$, la propiedad se verifica.
- $\pi_z = Abort_j$. Al ejecutarse esta acción, los conjuntos $set\_waiters_j$ y $setPred_j$ pasan a estar vacíos.

■

Las siguientes tres propiedades permiten establecer una correspondencia entre los posibles valores de *state* (variable de estado del autómata G) y *status.alg* (variable de estado del autómata A).

Propiedad B.3.3. Sea s un estado alcanzable de S. Se verifica que $\forall\ i \in \mathcal{N}$: $s.state_i = active \Leftrightarrow s.status.alg_i = active$.

Demostración: El estado inicial, s_0, verifica que $\forall\ i \in \mathcal{N}$: $s_0.state_i = active$ y $s_0.status.alg_i = active$, por lo que la propiedad se cumple. Las acciones que afectan a las variables de la propiedad son:

- $\pi_z = StartAddArc_i(j)$. Las variables implicadas cambian por los efectos de la acción: $s_z.state_i = blocked$ y $s_z.status.alg_i \in \{candidate,\ blocked\}$.

- $\pi_z = EndDelArc_i(j)$. Los efectos de la acción hacen que $s_z.status.alg_i = active$ y $s_z.state_i = active$.

- $\pi_z = Abort_i$. Al ejecutarse la acción, $s_z.status.alg_i = aborted$ y $s_z.state_i = aborted$.

- $\pi_z = initiate_i$ o $\pi_z \in \{rcvALG_i(j,\ m)\colon m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVS_i(j,\ m)\colon m \in M_{AVS}\}$ o $\pi_z \in \{rcvINF_i(j,\ m)\colon m \in M_{INF}\}$. Al ejecutarse cualquiera de estas acciones, puede cambiar $status.alg_i$ de tal forma que $s_z.status.alg_i \in \{dummy, candidate, victim\}$. Además, para que esos cambios estén habilitados, $s_{z-1}.status.alg_i \in \{blocked, candidate\}$. Considerando la hipótesis inductiva en esos casos, se asegura que $s_{z-1}.state_i \neq active$. Dado que la variable $state_i$ no es modificada por los efectos de estas acciones, la propiedad queda demostrada.

■

Propiedad B.3.4. Sea s un estado alcanzable de S. Se verifica que $\forall\ i \in \mathcal{N}$: $s.state_i = blocked \Leftrightarrow s.status.alg_i = \{blocked, dummy, candidate, victim\}$.

Demostración: En el estado inicial, s_0, se verifica la propiedad porque $\forall\ i \in \mathcal{N}$: $s_0.state_i = s_0.status.alg_i = active$. Las acciones que afectan a las variables de la propiedad son:

- $\pi_z = StartAddArc_i(j)$. Al ejecutarse la acción, $s_z.state_i = blocked$ y $s_z.status.alg_i \in \{blocked, candidate\}$.

- $\pi_z = EndDelArc_i(j)$. La ejecución de la acción hace que $s_z.state_i = active$ y $s_z.status.alg_i = active$.

- $\pi_z = Abort_i$. Por los efectos de la acción $s_z.status.alg_i = aborted$ y $s_z.state_i = aborted$.

- $\pi_z = initiate_i$ o $\pi_z \in \{rcvALG_i(j,\ m)\colon m \in M_{ALG}\}$ o $\pi_z \in \{rcvAVS_i(j,\ m)\colon m \in M_{AVS}\}$ o $\pi_z \in \{rcvINF_i(j,\ m)\colon m \in M_{INF}\}$. Entre los efectos posibles de estas acciones están los que consisten en cambiar el estado del nodo i, haciendo que $s_z.status.alg_i \in \{dummy, candidate, victim\}$. En todas estas situaciones, $s_{z-1}.status.alg_i \in \{blocked, candidate\}$. Por tanto, la hipótesis inductiva indica que $s_{z-1}.state_i = blocked$. Como ninguna de estas acciones cambia el valor de $state_i$, la propiedad se cumple.

■

Propiedad B.3.5. Sea s un estado alcanzable de S. Se verifica que $\forall\ i \in \mathcal{N}$: $s.state_i = aborted \Leftrightarrow s.status.alg_i = aborted$.

Demostración: Por reducción al absurdo. Se supone que $s_z.status.alg_i \neq aborted$, siendo $s_z.state_i = aborted$. La única acción que permite que la variable $state_i$ sea *aborted* es $Abort_i$. Si la acción $Abort_i$ está habilitada entonces $s_{z-1}.state_i \neq aborted$, esto es, $s_{z-1}.state_i \in \{active, blocked\}$. En caso de que $s_{z-1}.state_i = active$, la propiedad B.3.3 asegura que $s_{z-1}.status.alg_i = active$ y la accción $Abort_i$ no podría ser ejecutada. Si $s_{z-1}.state_i = blocked$, la propiedad B.3.4 indica que $s_{z-1}.status.alg_i \in \{blocked, dummy, candidate, victim\}$. Para los tres primeros estados, la acción $Abort_i$ no estaría habilitada y para el estado *victim*, los efectos de la acción hacen que $s_z.status.alg_i = aborted$. Considerando el sentido contrario de la implicación de la propiedad, se asume que $s_z.status.alg_i = aborted$ y $s_z.state_i \neq aborted$. Sólo la acción $Abort_i$ provoca $status.alg_i$ sea *aborted*. La condición de habilitación de esta acción implica que $s_{z-1}.status.alg_i = victim$ y, aplicando la propiedad B.3.4, se obtiene que $s_{z-1}.state_i = blocked$. La ejecución de $Abort_i$ hace que $state_i$ pase a ser *aborted*. ■

Apéndice C

Notación

C.1. Conjuntos

Símbolo	Significado	Ejemplo
$\{\}$	conjunto	$\{1,2,\dots\}$
$\emptyset$	conjunto vacío	$\emptyset = \{\ \}$
$\mathbb{N}$	conjunto de los números naturales	$\mathbb{N} = \{1,2,3,\dots\}$
$\mathbb{Z}$	conjunto de los números enteros	$\mathbb{Z} = \{0,1,\text{-}1,2,\text{-}2,\dots\}$
$\in$	pertenece, es un elemento de ...	$2 \in \mathbb{N}$
$\notin$	no pertenece, no es un elemento de ...	$\text{-}2 \notin \mathbb{N}$
$\subseteq$	está contenido, es un subconjunto de ...	$\mathbb{N} \subseteq \mathbb{Z}$
$\cup$	unión	$\{1,2\} \cup \{3\} = \{1,2,3\}$
$\cap$	intersección	$\{1,2,3\} \cap \{1,2\} = \{1,2\}$
$\setminus$	menos	$\{1,2,3\} \setminus \{1\} = \{2,3\}$
$\times$	producto cartesiano	$(x,\ y) \subseteq \mathbb{N} \times \mathbb{Z}$ $(x \in \mathbb{N},\ y \in \mathbb{Z})$

C.2. Lógica

Símbolo	Significado	Ejemplo
$\equiv$	equivale a, se define como	$\emptyset \equiv \{\ \}$
$\forall$	para todo	$\forall\ x \in \mathbb{N}$ equivale a $x \in \{1,2,3,\dots\}$
:	se verifica que	$\forall\ x \in \mathbb{N} : x - x = 0$
$\mid$	tal que	$\forall\ x \in \mathbb{N} \mid x < 3$ equivale a $x \in \{1,2\}$
$\exists$	para algún, existe...	$\exists\ x \in \mathbb{Z} \mid x \geq 0$ equivale a $x \in \{0,1,2,\dots\}$
$\nexists$	no existe...	$\nexists\ x \in \mathbb{N} \mid x < 0$ equivale a $\emptyset$
$\Rightarrow$	IMPLICA (si... entonces...)	$x \in \mathbb{N} \Rightarrow x > 0$
$\Leftrightarrow$	EQUIVALE (si y sólo si... entonces...)	$x > y \Leftrightarrow y < x$
$\wedge$	Y	$x \in \mathbb{N} \wedge x < 2 \Rightarrow x = 1$
$\vee$	O	$x \in \mathbb{Z} \Rightarrow x \geq 0 \vee x < 0$

C.3. Secuencias

Símbolo	Significado	Ejemplo
γ, γ_1, γ_2	secuencias o cadenas indeterminadas	$\gamma = 1,1,2$
ε	secuencia vacía	$\gamma \neq \varepsilon$
$\cdot$	concatenador de secuencias	$\gamma_1 \cdot \gamma_2$

Bibliografía

[1] G. Gardarin. *Une Solution au Probleme du Verrou Mortel in Partage Simultaneé de Fichiers*. PhD thesis, L'Université de Paris VI, Paris, 1974.

[2] E. G. Coffman, M. J. Elphick, and A. Shoshani. Systems Deadlocks. *ACM Computing Surveys*, 3(2):67–78, 1971.

[3] P. Brinch. *Operating System Principles*. Prentice-Hall, London, 1973.

[4] D. J. Rykpa and A. P. Lucido. Deadlock detection and avoidance for shared logical resources. *IEEE Transactions on Software Engineering*, SE-5(5):465–471, 1979.

[5] S. S. Isloor and T. A. Marsland. The deadlock problem: An overview. *IEEE Computer*, 13(9):58–78, 1980.

[6] R. C. Holt. *On Deadlock in Computer Systems*. PhD thesis, Cornell University, 1971.

[7] R. C. Holt. Comments on prevention of system deadlocks. *Communications of the ACM*, 14(1):36–38, 1971.

[8] R. C. Holt. Some deadlock properties on computer systems. *ACM Computing Surveys*, 4(3):179–196, 1972.

[9] J. W. Havender. Avoiding deadlock in multistasking systems. *IBM Systems Journal*, 7(2):74–84, 1968.

[10] A. Silberschatz, J. Peterson, and P. B. Galvin. *Sistemas Operativos. Conceptos fundamentales*. Addison Wesley Iberoamericana, Wilmington, Delaware, 1994.

[11] C. A. R. Hoare. *Towards a Theory of Parallel Programming*. International Seminar of Operating Systems Techniques. Academic Press, 1972.

[12] J. H. Howard. Mixed solutions for the deadlock problem. *Communications of the ACM*, 16(7):427–430, 1973.

[13] E. W. Dijkstra. The structure of the-multiprogramming system. *Communications of the ACM*, 11(5):341–346, 1968.

[14] E. Knapp. Deadlock detection in distributed databases systems. *ACM Computing Surveys*, 19(4):303–328, 1987.

[15] F. Fariña. *Una Solución al Problema del Interbloqueo en Sistemas Distribuidos con Modelo AND de Petición de Recursos*. PhD thesis, Universidad Pública de Navarra, Departamento de Matemática e Informática, 1997.

[16] J. Villadangos. *Un Algoritmo Distribuido para la Resolución del Interbloqueo en el Modelo OR*. PhD thesis, Universidad Pública de Navarra, Departamento de Automática y Computación, 1997.

[17] G. Bracha and S. Toueg. *A Distributed Algorithm for Generalized Deadlock Detection*. Technical Report TR 83-558, Dept. Computer Science, Cornell University, 1983.

[18] E. W. Dijkstra. *Cooperating sequential processes*. F. Genuys, Programming Languages: NATO Advance Study Instute, Academic Press, 1968.

[19] E. W. Dijkstra. *The Bankiers Algorithme*. Technical Report EWD166, University of Eindoven, Dept. Mathematics, 1965.

[20] J. E. Murphy. Resource allocation with interlock detection in a multitask system. In *AFIPS, FJCC*, volume 33, pages 1168–1176, 1968.

[21] A. N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12(7):373–385, 1969.

[22] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice-Hall Inc. Englewood Cliffs, New Jersey, 1995.

[23] C. Beeri and R. Obermarck. A resource class independent deadlock detection algorithm. In *7th International IEEE Conference on Very Large Databases, Cannes, France*, pages 166–178, 1981.

[24] A. Shoshani and E. G. Coffman. Sequencing tasks in multi-process, multiple resouce systems to avoid deadlocks. In *11th Annual IEEE Symposium on Switching and Automata Theory*, pages 225–235, 1970.

[25] E. M. Gold. Deadlock prediction: Easy and difficult cases. *SIAM Journal Computer*, 7(3):320–336, 1978.

[26] Y. Sugiyama, T. Araki, J. Okui, and T. Kasani. Complexity of the deadlock avoidance problem. *Transactions of the Institute of Electronics, Communications and Engineering of Japan*, J60-D(4):251–258, 1977.

[27] R. Devillers. Game interpretation of the deadlock avoidance problem. *Communications of the ACM*, 20(10):741–745, 1977.

[28] W. W. Chu and G. Ohlmacher. Avoiding deadlock in distributed databases. In *ACM Annual Conference*, pages 156–160, 1974.

[29] J. Gray. Notes on database operatting systems. *Operating Systems: An Advanced Course. Lecture Notes in Computer Science*, 60:393–481, 1978.

[30] Y. Eguchi and T. Yoshinaga. Deadlock detection algorithm with level number. *Systems and Computers*, 17(11):42–50, 1986.

[31] D. B. Lomet. A practical deadlock avoidance algorithm for database systems. In *International Conference on Management of Data (ACM SIGMOD)*, pages 122–127, 1977.

[32] D. D. Chamberlin, R. F. Boyce, and I. L. Traiger. Deadlock-free scheme for resource locking in data-base environment. In *IFIP 74*, pages 340–343, 1974.

[33] P. F. King and A. J. Collmeyer. Database sharing-and efficient method for supporting concurrent processes. In *1974 National Computer Conference*, volume 42, 1974.

[34] J. L. Peterson and A. Silberschatz. *Operating System Concepts*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.

[35] M. G. Gouda. Closed covers: To verify progrress for communicating finite state machines. *IEEE Transactions on Software Enginnering*, SE-10(6):846–855, 1984.

[36] M. G. Gouda and C. K. Chang. Proving liveness for networks of communicating finite state machines. *ACM Transactions on Programming Languages and Systems*, 8(1):154–182, 1986.

[37] A. Chung and D. P. Sidhu. On conditions for defining a closed cover to verify progress for communicating finite state machines. *IEEE Transactions on Software Engineering*, 15(1):1491–1494, 1989.

[38] J. Cheng. A classification of tasking deadlocks. *Ada Letters*, 10(5):110–127, 1990.

[39] K. S. Bagga and F. W. Owens. Some graph-theoretic properties of deadlocks and traps in petri nets. In *Graph Theory, Combinatorics and Applications*, volume 1, pages 19–28.

[40] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, Com-92:11–18, 1981.

[41] W. J. Rally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[42] J. M. Jaffe and M. Sidi. Distributed deadlock resolution in store-and-forward networks. *Algorithmica*, (4):417–436, 1989.

[43] V. C. Barbosa. Strategies for the prevention of communication deadlocks in distributed parallel programs. *IEEE Transactions on Software Engineering*, 16(11):1311–1316, 1990.

[44] W. K. Ng and C. V. Ravishankar. On-line detection and resolution of communication deadlocks. In *27th Annual Hawaii International Conference on System Sciences*, pages 524–533, 1994.

[45] J. Duato. A new theory of deadlock-free adaptative routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[46] M. Krishnamurthy, A. Basavatia, and S. Thallikar. Deadlock detection and resolution in simulation models. In *1994 Winter Simulation Conference*, pages 708–715, 1994.

[47] Z. MengChu. Deadlock avoidance schemes in a distributed robotic system: Petri net modeling and analysis. *Journal of Robotic Systems*, 12(3):177–187, 1995.

[48] J. A. Stankovic. A perspective on distributed computer systems. *IEEE Transactions on Computers*, C-33(12):1102–1125, 1984.

[49] M. D. Schroeder. *A state-of-the-art distributed system: computing with BOB*. ACM Press, Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[50] L. Lamport and N. Lynch. Distributed computing: Models and methods. In *Handbook of Theoretical Computer Science*, pages 1157–1199. J. van Leeawen, Elsevier Science Publisher B. V, 1990.

[51] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall Inc., Upper Saddle River, NJ, USA, 1985.

[52] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York Inc., Secaucus, NJ, USA, 1982.

[53] K. M. Chandy. *Parallel Program Design: a Foundation*. Addison-Wesley, Boston, MA, USA, 1988.

[54] N. Lynch and M. R. Tuttle. An Introduction to Input/Output Automata. *CWI Quaterly*, 2(3):219–246, 1989.

[55] M. Singhal. Deadlock detection in distributed systems. *IEEE Computer*, pages 37–48, 1989.

[56] A. D. Kshemkalyani and M. Singhal. Invariant-based verification of a distributed deadlock detection and resolution. *IEEE Transactions on Software Engineering*, 17(8):789–799, 1991.

[57] B. Goldman. *Deadlock Detection in Computer Networks*. Technical Report, Massachusets Institute of Technology, MIT, 1977.

[58] J. N. Gray. *A Discussion of Distributed Systems*. Res. Rep. RJ2699 (34394). IBM Research Division, 1979.

[59] M. Stonebraker. Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. *IEEE Transactions on Software Engineering*, 5(3):188–194, 1979.

[60] G. S. Ho and C. V. Ramamoorthy. Protocols for Deadlock Detection in Distributed Database Systems. *IEEE Transactions on Software Engineering*, 8(6):554–557, 1982.

[61] V. D. Gligor and S. H. Shattuck. On deadlock detection in distributed systems. *IEEE Transactions on Software Engineering*, SE-6(5):435–440, 1980.

[62] D. A. Menasce and R. R. Muntz. Locking and deadlock detection in distributed data bases. *IEEE Transactions on Software Engineering*, 5(3):195–202, 1979.

[63] J. R. Jaggannathan and R. Vasudevan. A distributed deadlock detection and resolution scheme: Performance study. In *3rd International Conference on Distributed Computing Systems*, pages 496–501, 1982.

[64] S. S. Isloor and T. A. Marsland. An effective on-line deadlock detection technique for distributed database management systems. In *2nd IEEE International Computer Software and Applications Conference*, pages 283–288, 1978.

[65] R. Obermarck. Distributed deadlock detection algorithm. *ACM Transactions on Databases Systems*, 7(2):187–208, 1982.

[66] K. M. Chandy, J. Misra, and L. M. Haas. Distributed deadlock detection. *ACM Transactions on Computer Systems*, 1(2):144–156, 1983.

[67] K. Sugihara et al. A distributed algorithm for deadlock detection and resolution. In *4th Symposium on Reliablity in Distributed Software and Database Systems*, pages 169–176, 1984.

[68] D. P. Mitchell and M. J. Merrit. A distributed algorithm for deadlock detection and resolution. In *3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 282–284, 1984.

[69] G. Bracha and S. Toueg. A distributed algorithm for generalized deadlock detection. In *3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 285–301, 1984.

[70] M. Sinha and N. Natarajan. A priority based distributed deadlock detection algorithm. *IEEE Transactions on Software Engineering*, SE-11(1):67–80, 1985.

[71] A. N. Choudhary, W. H. Kohler, J. A. Stankovic, and D. Towsley. Correction to 'a modified priority based algorithm for distributed deadlock detection and resolution'. *IEEE Transactions on Software Engineering*, 15(12):1644, 1989.

[72] D. J. Badal. The distributed deadlock detection algorithm. *ACM Transactions on Computer Systems*, 4:320–337, 1986.

[73] J. R. González de Mendívil, J. M. Bernabeu-Auban, A. Demaille, and J. R. Garitagoitia. Correctness of a distributed deadlock resolution algorithm for the single request model. In *3rd Euromicro Workshop in Parallel and Distributed Processing*, pages 254–261, 1995.

[74] Y. M. Kim, T. H. Lai, and N. Soundarajan. Efficient distributed deadlock detection and resolution using probes, tokens and barriers. In *1997 Inernational Conference on Parallel and Distributed Systems*, pages 584–591, 1997.

[75] B. Wang, Y. Goto, and J. Cheng. Queue operation related tasking deadlosks in ada 2012 programs. *ADA Letters ACM*, 34(2):9–25, 2014.

[76] D. Lefebvre. Deadlock-free scheduling for timed pretri net models combined with mpc and backtracking. In *13th International Workshop on Discrete Event Systems*, pages 466–471, 2016.

[77] Y. Yang and H. Hu. Distributed deadlock avoidance in atomated manufacturing systems with forward conflict free structures using petri nets. In *2016 European Control Conference (ECC)*, pages 2345–2352, 2016.

[78] M. Zhao and M. Uzam. A suboptimal deadlock control policy for designing non-blocking supervisors in flexible manufacturing systems. *Information Sciences - Elsevier*, 388-389:135–153, 2017.

[79] M. Foumani, I. Gunawan, and K. Smith-Miles. Resolution of deadlocks in a robotic cell scheduling problem with post-process inspection system: Avoidance and recovery scenarios. In *2015 IEEE IEEM*, pages 1107–1111, 2015.

[80] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen. Deadlocks in datacenter networks: Why do they form and how to avoid them. In *2016 HotNets-XV ACM*, pages 92–98, 2016.

[81] A. Shpiner, E. Zahavi, V. Zdornov, T. Anker, and M. Kadosh. Unlocking credit loop deadlocks. In *2016 HotNets-XV ACM*, pages 85–91, 2016.

[82] H.-M. Chou, Y.-C. Chen, K.-H. Yang, J. Tsao, S.-C. Chang, W.-B. Jone, and T.-F Chen. High-performance deadlock-free id assignment for advanced interconnect protocols. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):1169–1173, 2016.

[83] A. Shao, D. Wang, and H. Wang. Pull-off buffer: Borrowing cache space to avoid deadlock for fault-tolerant noc routing. volume 24, pages 464–471, 2016.

[84] Z. D. Luo, R. Das, and Y. Qi. Multicoresdk: A practical and efficient deadlock detector for real-world applications. In *4th IEEE International Conference on Software Testing, Verification and Validation*, pages 309–318, 2011.

[85] M. Samak and M. K. Ramarathan. Multithreaded test synthesis for deadlock detection. In *Conference on Object-oriented Program Systems and Applications Companion*, pages 473–489, 2014.

[86] Y. Cai, S. Wu, and W. K. Chan. Conlock: A constraint-based approach to dynamic checking on deadlock in multithreaded programs. In *ICSE ACM*, pages 491–502, 2014.

[87] T. Cogumbreiro, R. Hu, F. Martins, and N. Yoshida. Dynamic deadlock verification for general barrier synchronization. In *2015 PPoPP ACM*, pages 150–160, 2015.

[88] Z. Lin, H. Zhong, Y. Chen, and J. Zhao. Lockpeeker: Detecting latent locks in java apis. In *2016 ASE ACM*, pages 368–378, 2016.

[89] N. Ng and N. Yoshida. Static deadlock detection for concurrent go by global session graph synthesis. In *2016 CC ACM*, pages 174–184, 2016.

[90] N. Kobayashi and C. Laneve. Deadlock analysis of unbounded process networks. *Information and Computation - Elsevier*, 250:48–70, 2017.

[91] S. Reveliotis and Z. Fei. Robust deadlock avoidance for sequential allocation systems with resource outages. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 864–871, 2016.

[92] P. Li, J. Beard, and J. Buhler. Deadlock-free buffer configuration for stream computing. In *PMAM ACM*, pages 164–169, 2015.

[93] V. V. Barbosa, A. Carneiro, F. Protti, and U. S. Souza. Deadlock models in distributed computation: Foundations, design, and computational complexity. In *2016 SAC ACM*, pages 538–541, 2016.

[94] W. Lu, Y. Yang, L. Wang, W. Xing, and X. Che. A leader election based detection algorithm in distributed system. In *SCTDCP ACM*, pages 1–19, 2016.

[95] M. Prieto, J. Villadangos, F. Fariña, and A. Córdoba. An $\mathcal{O}(n)$ distributed deadlock resolution algorithm. In *14th Euromicro Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society*, pages 136–143, 2006.

[96] M. Castillo, F. Fariña, and A. Córdoba. A Dynamic Deadlock Detection/Resolution Algorithm with Linear Message Complexity. In *20th Euromicro International Conference on Parallel Distributed and Network-Based Processing, IEEE Computer Society*, pages 175–179, 2012.

[97] G. LeLann. Distributed systems: Towards a formal approach. In *Information Processing*, pages 155–160, North-Holland, 1977. B. Gilchrist.

[98] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, New York, 1994.

[99] E. J.-H. Chang and R. Roberts. An improved algorithm for decentralized extrema finding in circular arrangements of processes. 22:281–283, 1979.

[100] D. S. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processes. *Communication ACM*, 23:627–628, 1980.

[101] G. L. Peterson. An $\mathcal{O}(n\log n)$ unidirectional algorithm for the circular extrema problem. *ACM Transactions Programming Language System*, 4:758–762, 1982.

[102] D. Dolev, M. Klawe, and M. Rodeh. An $\mathcal{O}(N\log N)$ unidirectional distributed algorithm for extrema-finding in a circle. *Algorithms*, 3:245–260, 1982.

[103] H. L. Bodlaender. A better lower bound for didtributed leader finding in bidirectional asynchronous rings of processors robust deadlock avoidance for sequential allocation systems with resource outages. *Information Processing Letters*, 25:287–290, 1988.

[104] J. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum finding algorithms. *ACM*, 31:905–918, 1984.

[105] G. Tel. *Topics in Distributed Algorithms*, volume 1 of *Cambridge Int. Series on Parallel Computation*. Cambridge University Press, New York, 1991.

[106] E. Korach, S. Kutten, and S. Moran. A modular tecnique for the design of efficient leader finding algorithms. *ACM Transactions Programming Language System*, 12:84–101, 1990.

[107] C. L. Michael, A. M. Teresa, and B. W. Douglas. Election in a complete network with a sense of direction. *Information Processing Letters*, 28(6):327, 1988.

[108] T. Masuzawa, N. Nishikawa, K. Hagihara, and N. Tokura. Optimal fault tolerant distributed algorithms for election in complete networks with a global sense of direction. pages 171–182, 1989.

[109] G. Shingh. Leader election in complete networks. *SIAM Journal on Computing*, 6(3):772–785, 1997.

[110] G. Shingh. Efficient leader election using sense of direction. *Distributed Computing*, 10(3):159–165, 1997.

[111] J. Villadangos, A. Córdoba, F. Fariña, and M. Prieto. Efficient Leader Election in Complete Networks. In *13th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society*, pages 136–143, 2005.

[112] M. Prieto. *Un Algoritmo Óptimo para la Detección y Resolución de Interbloqueos en Sistemas Distribuidos*. PhD thesis, Universidad Pública de Navarra, Departamento de Ingeniería Matemática e Informática, 2007.

[113] M. Castillo, F. Fariña, A. Córdoba, and J. Villadangos. A modified $\mathcal{O}(n)$ leader election algorithm for complete networks. In *15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society*, pages 189–198, 2007.

[114] M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. Master's thesis, Massachusets Institute of Technology MIT, Dept. of Electrical Engineering and Computer Science, 1987.

[115] K. M. Chandy and J. Misra. A distributed algorithm for detecting resource deadlocks in distributed systems. In *1st ACM Symposium on Principles of Distributed Computing*, pages 157–164, 1982.

[116] C. Berge. *Graphs*. Elsevier Science Publishers, North Holland, 1985.

[117] A. Fekete, N. Lynch, Y. Mansour, and J. Spinelli. The Impossibility of Implementing Reliable Communication in the Face of Crashes. *Journal of the ACM*, 40(5):1087–1107, 1993.

[118] P. Li and B. McMillin. *Fault Tolerant Distributed Deadlock Detection/Resolution*. Technical Report CSC-92-04, Department of Computer Science, University of Missoury at Rolla, 1992.

[119] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, Boston, MA, USA, 1975.

[120] L. Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. John Wiley & Sons, Boston, MA, USA, 1976.

[121] A. N. Choudhary, W. H. Kohler, J. A. Stankovic, and D. Towsley. A modified priority based algorithm for distributed deadlock detection and resolution. *IEEE Transactions on Software Engineering*, 15(1):10–17, 1989.

[122] J. R. González de Mendívil, F. Fariña, A. Demaille, and J. R. Garitagoitia. A simple distributed deadlock solver. In *21st Euromicro Conference*, pages 587–594, 1995.

[123] University of Southern California/Information Sciences Institute. UDP, User Datagram Protocol. https://tools.ietf.org/html/rfc768, 1980.

[124] University of Southern California/Information Sciences Institute. TCP, Transmission Control Protocol. https://tools.ietf.org/html/rfc793, 1981.

[125] C. A. Fidge. Logical time in distributed computing systems. *IEEE Computer*, pages 28–33, 1991.

[126] M. Castillo, F. Fariña, and A. Córdoba. A dynamic $\mathcal{O}(n)$ deadlock detection and resolution algorithm in process networks. In *4th International Conference on Computer Science and Software Engineering (CSSE) - CiSE*, pages 1712–1716, 2011.

[127] M. Castillo, F. Fariña, and A. Córdoba. Deadlock in process networks: a dynamic detection and resolution. In *5th International Conference on Signal Processing and Communication Systems, ICSPCS11*, page 9, 2011.

[128] N. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.

[129] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. Timed I/O automata: A mathematical framework for modeling and analyzing real-time systems. In *24th IEEE International Real-Time Systems Symposium*, pages 166–177, 2003.

[130] S. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviours of probabilistic i/o automata. *Theorical Computer Science*, 176(1-2):1–38, 1997.